

Implementation

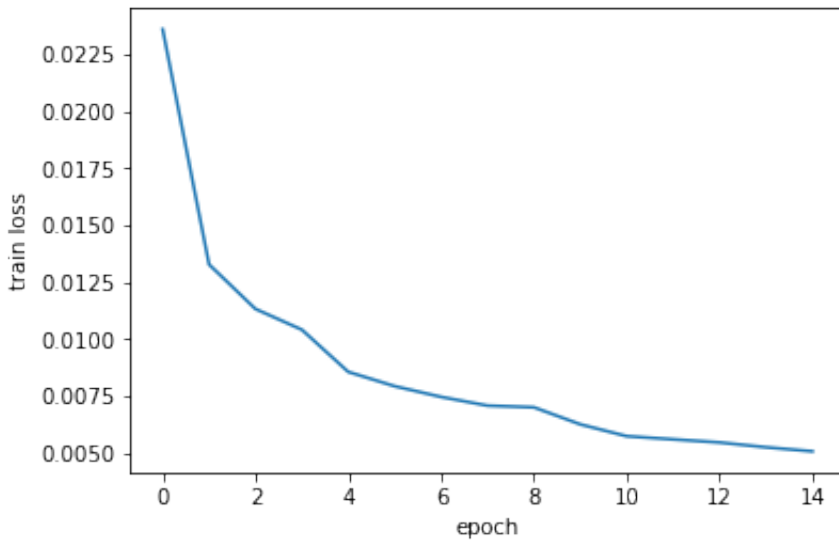
1. Use the function `generate_training_data_set`, function `generate_testing_data_set`, and class `TinyImageNet` in the `utils.py` to generate train datasets, test datasets and load two datasets respectively. The sampling strategy for training datasets are:
 - Loop through the image database, and treat each image as a query.
 - For the positive image, sample uniformly at random in the same folder of the query image, while sample uniformly at random in the different folder of the query image for the negative image
 - In each epoch, re-sample the whole triplets.
2. Due to the testing cost, I trained `resnet50` for 15 epochs without checking the training accuracy and testing accuracy and only record train loss at save the model for each epoch.
3. After finish training for a certain number of epochs, I pass images from the `val` folder to `test.py` to get the test accuracy.
4. For report purpose, use the `val.py`

Results

- Final **Testing Accuracy: 50.3%: Training Accuracy: 51.1%**
- Training Loss is given below:

```
import pickle
import matplotlib.pyplot as plt
%matplotlib inline
from PIL import Image
query_list, mytop10, mybottom10, loss = pickle.load(open("./myfinal.p", 'rb'))
```

```
plt.plot(loss)
plt.xlabel("epoch")
plt.ylabel("train loss")
plt.show()
```



Query results

Distance is given at the title in the format of `label/distance`. Here the label is the folder name for each class.

The first two rows are the result for the top10 while the rest two columns are for the bottom 10.

```
def show_query(query_idx, flag="top"):
    if flag == "top":
        fig, axs = plt.subplots(2,6, figsize=(18, 6))
        fig.subplots_adjust(hspace = .5, wspace=.001)
        count = 0
        for i in [0,1]:
            for j in [0,1,2,3,4,5]:
                if i == 0 and j == 0:
                    img = Image.open(query_list["path"][query_idx])
                    label = query_list["label"][query_idx]
                    axs[i,j].imshow(img)
                    axs[i,j].set_title(label)
                    axs[i,j].axis('off')
                else:
                    if (count <= 9):
                        img = Image.open(mytop10["path"][query_idx][count])
                        label = mytop10["label"][query_idx][count]
                        distance = mytop10["distance"][query_idx][count]
                        axs[i,j].imshow(img)
                        axs[i,j].set_title(label+ "/" +str(str(distance).split("(")[1
```

```

].split(" ")[0]))
        axs[i,j].axis('off')
        count += 1
    else:
        axs[i,j].axis('off')
else:
    fig, axs = plt.subplots(2,6, figsize=(18, 6))
    fig.subplots_adjust(hspace = .5, wspace=.001)
    count = 0
    for i in [0,1]:
        for j in [0,1,2,3,4,5]:
            if i == 0 and j == 0:
                img = Image.open(query_list["path"][query_idx])
                label = query_list["label"][query_idx]
                axs[i,j].imshow(img)
                axs[i,j].set_title(label)
                axs[i,j].axis('off')
            else:
                if (count <= 9):
                    img = Image.open(mybottom10["path"][query_idx][count])
                    label = mybottom10["label"][query_idx][count]
                    distance = mybottom10["distance"][query_idx][count]
                    axs[i,j].imshow(img)
                    axs[i,j].set_title(label+ "/" +str(str(distance).split("(")[1
].split(" ")[0]))
                    axs[i,j].axis('off')
                    count += 1
                else:
                    axs[i,j].axis('off')

```

```

show_query(0,flag="top")
show_query(0,flag="bottom")

```

['n03444034']



n03444034/6.2222



n03444034/6.3626



n03444034/6.3730



n03444034/6.4214



n03444034/6.6379



n03444034/6.6966



n03444034/6.7117



n03444034/6.7552



n03444034/6.7796



n03444034/6.7805



['n03444034']



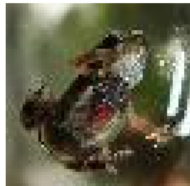
n03977966/22.8941



n02113799/22.8320



n01644900/22.7037



n02129165/22.3531



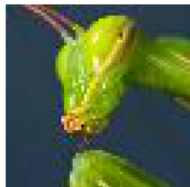
n02843684/22.3430



n03250847/22.1167



n02236044/21.9441



n04562935/21.8122



n03706229/21.7886



n02094433/21.7729



```
show_query(1,flag="top")
show_query(1,flag="bottom")
```

['n03837869']



n03837869/6.2822



n03837869/6.3536



n03837869/6.4110



n03837869/6.4806



n03837869/6.6436



n03837869/6.6447



n03837869/6.6888



n03837869/6.7248



n03837869/6.7361



n03837869/6.7419



['n03837869']



n01784675/22.8983



n02883205/22.7688



n07614500/22.5721



n04265275/22.2424



n03447447/22.2421



n03544143/22.0124



n03126707/22.0029



n02808440/21.8220



n04371430/21.6890



n02123394/21.6674



```
show_query(2,flag="top")
show_query(2,flag="bottom")
```

['n02814533']



n02814533/6.3278



n02814533/6.3733



n02814533/6.4791



n02814533/6.4843



n02814533/6.6183



n02814533/6.6709



n02814533/6.6943



n02814533/6.6949



n02814533/6.7144



n02814533/6.7327



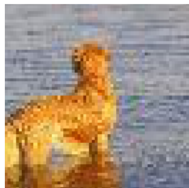
['n02814533']



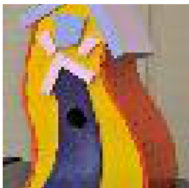
n03854065/22.8804



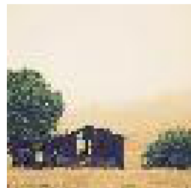
n02099601/22.7880



n02843684/22.5973



n02793495/22.2569



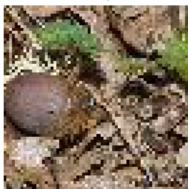
n07920052/22.2335



n07583066/22.0061



n12267677/21.9925



n02823428/21.8126



n02509815/21.7152



n04133789/21.6818

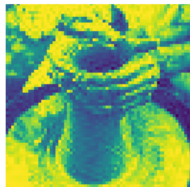



```
show_query(3,flag="top")
show_query(3,flag="bottom")
```

['n03992509']



n03992509/6.3144



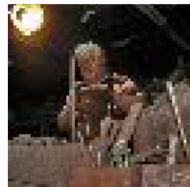
n03992509/6.3412



n03992509/6.3433



n03992509/6.4759



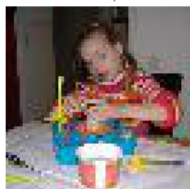
n03992509/6.6313



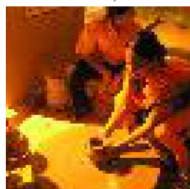
n03992509/6.7063



n03992509/6.7165



n03992509/6.7201



n03992509/6.7506



n03992509/6.7663



['n03992509']



n02281406/22.9290



n03770439/22.8895



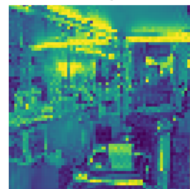
n03837869/22.7194



n02231487/22.4049



n02791270/22.3946



n03014705/22.1705



n02437312/22.0536



n02927161/21.8838



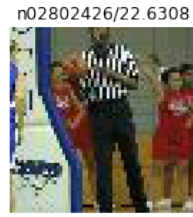
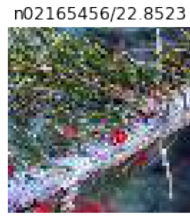
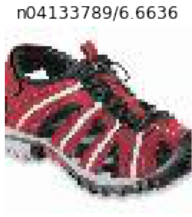
n02233338/21.8089



n07875152/21.8076



```
show_query(4,flag="top")
show_query(4,flag="bottom")
```



Note: By the time I submitted the homework, I fails to finish this part for my trained resnet50, simply because I am running out of the time. So the queried figures and distances are for resnet18, which is trained for 10 epochs with testing accuray 49.8%. But the traning accuracy, testing accuracy and traning loss reported above are for resnet50.

Improvement

The improvement can be done by fining the sampling strategy. Since we are interested in the figures that most "relevant" to our query. We need to define a relevance score, as suggested by the paper to ensure we can get more positive samples for the given query image.

For every picture p_i , it has a class label c_i . We define the relevance for p_i and p_j as

$$r_{ij} = \begin{cases} \text{some pre-calculated number,} & c_i = c_j, i \neq j \\ 0, & c_i \neq c_j, i \neq j \end{cases}$$

Then we sample the positive sample p_i^+ for p_i with following probability from the category c_i

$$P(p_i^+) \propto \min\{T_i, r_{i,i^+}\},$$

where T_i is a non-negative threshold.

Also, there are two types of negative samples, `in-class-negative` and `out-class-negative`.

For `out-class-negative` p_i^- , just sample uniformly at random while for `in-class-negative`, using $P(p_i^-) \propto \min\{T_i, r_{i,i^-}\}$