IE 534: Deep Learning, Fall 2018

Homework 5: Deep Ranking using ResNet on Tiny ImageNet

Lecturer: Justin Sirignano *Submitted by: Rachneet Kaur, Oct. 22th, 2018*

Implementation

The objective is to **find similar images** to the the input 64×64 dimensional coloured query image belonging to the same class as that of the query image, out of the 200 classes. The Tiny ImageNet data set has 100,000 coloured training images with 500 images per each of the 200 classes and 10,000 images in the validation set of dimensions $3 \times 64 \times 64$.

- Some images in the training and test set were grey scaled so I used $PIL.Image.open(image\_query\_name).convert('RGB')$ from PIL library in Python, to convert them into three channel (RGB) coloured images.

Data Loading and Triplet Sampling

To load the data set, I defined a class *CustomDataset* and used *torch.utils.data.DataLoader* function in Pytorch. To sample query, positive and negative images respectively in each triplet, I wrote a function $create\_triplets()$ in my implementation.

- For each of the 100,000 images in the training set, I select out of 499 other images from the same class as that of the query image uniformly at random a positive image using $image\_positive\_name = np.random.choice(images)$.
- To sample a negative image, I select uniformly at random, an image from one of the other 199 classes than that of the query image.

In total, I sample 100,000 new triplets each epoch and shuffle their ordering as well to ensure randomness in sample generation. This $\_\_getitem\_\_()$ function in my implementation returns a tuple of three images as $triplet = (q\_image, p\_image, n\_image)$.
Since we also need to compute precision for training set and test set, I keep a flag *training* in my *CustomDataset* $\_\_init\_\_$ function.

- If $training = 1$, a $triplet = (q\_image, p\_image, n\_image)$ is returned.

- If $training = 2$, then a tuple with a training set image and it's corresponding label is returned.
- If $training = 3$, then a tuple with a validation set image and it's corresponding label is returned.

Hence,

- $\_\_init\_\_()$ function in my implementation of class $CustomDataset$ creates lists for triplets, shuffles it and training and validation images with their labels.
- I generate new triplets and shuffle them randomly while the training phase.
- $\_\_getitem\_\_()$ function in my implementation returns a tuple of triplet, a training image with it's label or a validation set image with it's label depending on the the task.
- While loading the triplets, to use transfer learning from the pretrained Resnet50 model, I re-size the images to 224×224 and normalize the data.
- Since we need to compute the feature embedding for each image, I have 4096 units for 4096-dimensional feature embedding in the output layer.
- $\_\_len\_\_()$ function in my implementation returns the length of triplet list or the validation set list depending on the task.

Model Architecture

I used the pretrained Resnet 50 on ImageNet data-set to load the initial weights for the network as $model = torchvision.models.resnet50(pretrained = True)$. Also, to fine-tune the model, I set the number of outputs as 4096, as the feature embedding dimension we want to have.
Since the model is pretraiend on ImageNet, with image size = 224×224 and Tiny ImageNet images are 64×64, we re-size the train and validation images to 224×224.
Once the pretrained model is loaded, I train it on the triplets sampled from the Tiny ImageNet dataset for 12 epochs. My training time is about 30 hours on Blue waters.
Hence,

- Change the number of outputs in the last layer of the pretrained network to 4096.
- To resize the images in the training and testing set, *transforms.Resize(size=(224, 224))* function is used in the data loading step for both training and testing images loader.
- I use the Stochastic Gradient Descent optimizer with learning rate 0.001 and momentum 0.9.

- For implementation, I trained the Resnet50 for 12 epochs to achieve the target accuracy of 53.34%.
- I used the batch size = 20 for sampling the triplets in the training phase.

The three images in each triplet are passed through the Resnet50 architecture to compute their corresponding feature embedding. These three feature embedding are fed to the loss function, to compute the triplet hinge loss, which is then propagated backwards to optimize the parameters for the model.

Optimizer and Loss function

I used the Stochastic Gradient Descent optimizer with momentum = 0.9, implemented by *torch.optim.SGD(model.parameters(), lr=LR, momentum = 0.9)* to minimize the triplet hinge loss function, implemented by *torch.nn.TripletMarginLoss(margin=1.0, p=2, reduction='elementwise_mean')*.

- For implementation, I used SGD optimizer to minimize the triplet margin loss function with learning rate $\alpha = 0.001$ and $\alpha = 0.0001$ after 7 epochs.
- I used *torch.nn.TripletMarginLoss(margin=1.0, p=2, reduction='elementwise_mean')* loss function inbuilt in Pytorch to calculate the triplet hinge loss, to be back propagated.

Results

Precision@30 results

The triple margin training loss after 12 epochs is 0.06.
I used nearest neighbours *NearestNeighbors* from *sklearn.neighbors* package in Python to compute the 30 nearest feature embedding in the training set as to the query images' feature embedding.
For calculating precision, I count the no. of correct class members out of these nearest 30 retrieved by the model and divide by 30, to calculate the precision@30 for one query image. Averaging this precision over all query images in the training set and test set gives the precision@30 scores for the training and the test sets respectively.
The Precision@30 for the training and test set are as follows:

- Training loss after 12 epochs = 0.06
- Precision@30 on the training set is = 53.34%

- Precision@30 on the validation set is = 70.15%
- The estimated time to train the model for 12 epochs is about 30 hours.

Plotting the loss function

Figure 1 plots the triple margin loss function for training the Resnet50 model for 12 epochs.

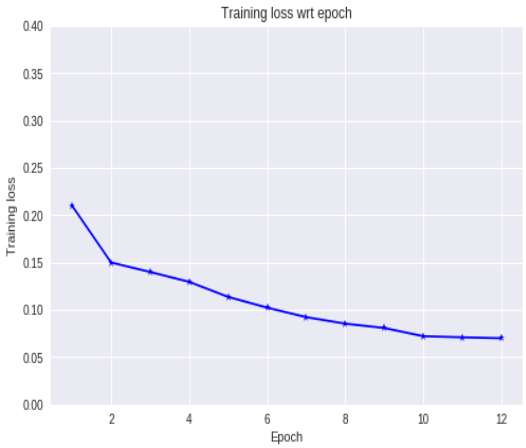


Figure 1: Plotting the training Loss

Qualitative Results: Top and Bottom 10 images:

For qualitative results, I use indices of $neighmin10 = NearestNeighbors(n\_neighbors = 10).fit(trained\_embeddings\_array)$ to compute the ten nearest neighbor embedding in the training set as to 5 randomly selected validation set images and last 10 indices of $neighmax10 = NearestNeighbors(n\_neighbors = 100000).fit(trained\_embeddings\_array)$ to compute the ten farthest neighbor embedding in the training set as to these validation set images.

I randomly select five validation set images and below are the retrieved top 10 and bottom 10 images with their **class labels and distances** from the queried image.
For validation image 1 (2), top 10 (3) and bottom 10 (4) images, with their **class labels and distances** from the validation image are as follows:

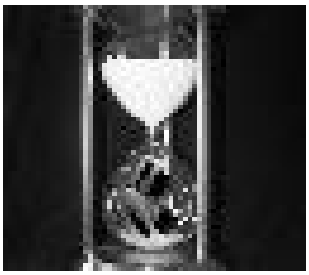
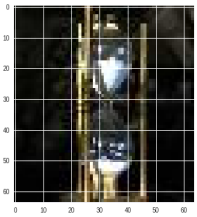
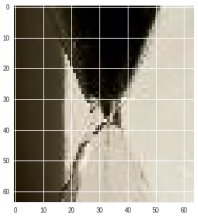


Figure 2: Validation Set Image 1: n03544143

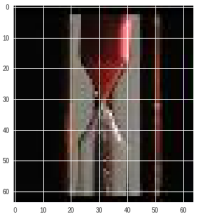
For validation image 2 (5), top 10 (6) and bottom 10 (7) images, with their class labels and distances from the validation image are as follows:



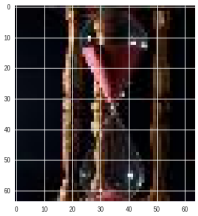
n03544143, Dis: 3.73



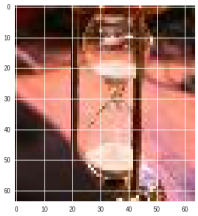
n03544143, Dis: 3.74



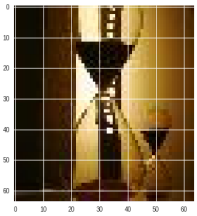
n03544143, Dis: 4.14



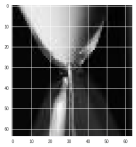
n03544143, Dis: 4.16



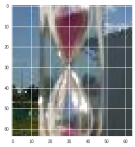
n03544143, Dis: 4.18



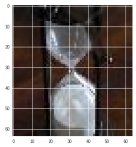
n03544143, Dis: 4.22



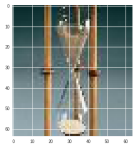
n03544143, 4.28



n03544143, 4.31



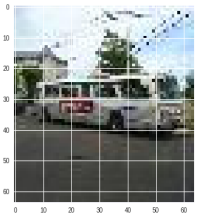
n03544143, 4.38



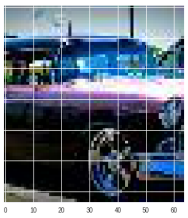
n0354414, 4.39

Figure 3: Top 10 images retrieved for Validation Set Image 1

For validation image 3 8, top 10 (9) and bottom 10 (10) images, with their class labels and distances from the validation image are as follows:

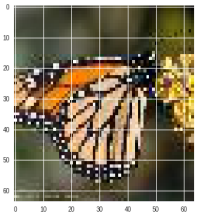


n04487081, Dis: 32.76



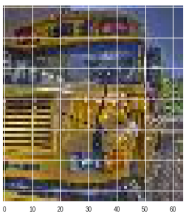
n02814533, Dis: 32.81

n03977966, Dis: 32.82



n02279972, Dis: 33.07

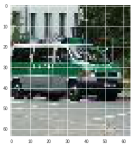
n02814533, Dis: 33.10



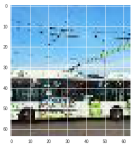
n04146614, Dis: 33.50

n03977966, 33.70

n04285008, 33.73



n03977966, 33.74



n04487081, 34.006

Figure 4: Bottom 10 images retrieved for Validation Set Image 1

For validation image 4 (11), top 10 (12) and bottom 10 (13) images, with their class labels and distances from the validation image are as follows:

Figure 5: Validation Set Image 2: n02666196

For validation image 5 (14), top 10 (15) and bottom 10 (16) images, with their class labels and distances from the validation image are as follows:

Improving the performance of the model

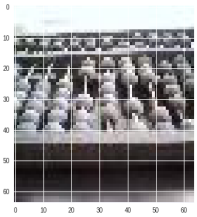
1. Multiple class N pair loss function instead of triplet loss

One way to improve the model performance is to replace the triplet loss function with a multiple class N pair loss function. To implement this approach, instead of passing one positive and one negative image with each query image, we pass multiple negative images in each iteration to revise the weights of the model.
This approach will improve the convergence rate of the model since instead of just learning from one negative example, we are using interaction of multiple negative example at each updating step. Note that this approach can be computationally expensive since we pass multiple negative images instead of just one image in each update. This problem can be overcome using some sophisticated improvements to the batch structure. The idea is if we pass negative examples, one from each class all at one update to the loss function, then the learning is quicker and improved.

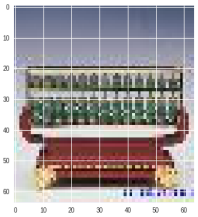
This idea is implemented in *[Sohn, Kihyuk. "Improved deep metric learning with multi-class n-pair loss objective." In Advances in Neural Information Processing Systems, pp. 1857-1865. 2016.]*
The loss function in this case for query image p, positive image p^+ and N negative images p_i^- for i = 1 to N, would be:

$$Loss(p, p^+, (p_i^-)_{i=1}^N) = log(1 + \sum_{i=1}^{N} e^{pp_i^- - pp^+}) \tag{1}$$

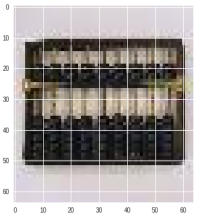
Another paper *[Chen, Weihua, Xiaotang Chen, Jianguo Zhang, and Kaiqi Huang. "Beyond triplet loss: a deep quadruplet network for person re-identification." In The IEEE Conference*



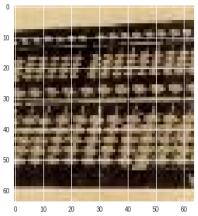
n02666196, Dis: 4.61



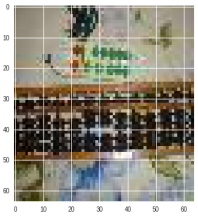
n02666196, Dis: 4.63



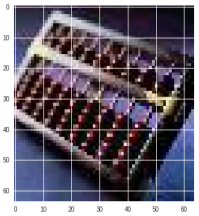
n02666196, Dis: 4.77



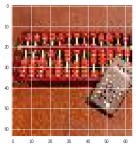
n02666196, Dis: 4.79



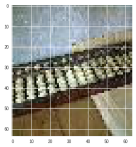
n02666196, Dis: 4.97



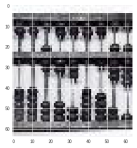
n02666196, Dis: 5.00



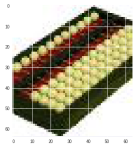
n02666196, 5.00



n02666196 5.05



n02666196, 5.10



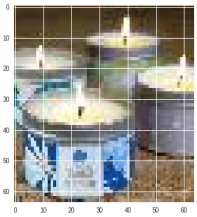
n02666196, 5.14

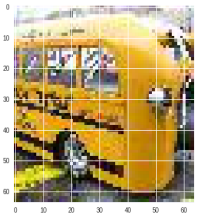
Figure 6: Top 10 images retrieved for Validation Set Image 2

on Computer Vision and Pattern Recognition (CVPR), vol. 2, no. 8. 2017.] use the quadruplet loss instead of the triplet loss to imporve the performance of the network.

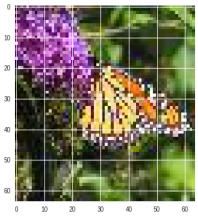
2. Weighted sampling with relevance score

In our implementation, we use uniform sampling of the images to construct triplets, instead the paper *[Wang, Jiang, Yang Song, Thomas Leung, Chuck Rosenberg, Jingbin Wang, James Philbin, Bo Chen, and Ying Wu. "Learning fine-grained image similarity with deep ranking." In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 1386-1393. 2014.]* uses weighted sampling of images where the weights are derived based on the relevance score. The sampling probability is proportional to the total and pairwise relevance scores of the images. Also, instead of just sampling out of class negative images, the paper also samples the in-class negative samples from the same class as that of the query image, but this in class negative sample is less relevant than the positive image we sample. The authors first calculate the total relevance score for each query image by summing the

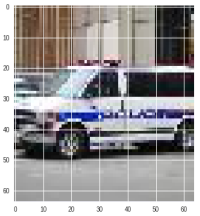

n02948072, Dis: 31.27

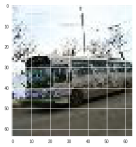

n04146614, Dis: 31.37


n03977966, Dis: 31.40

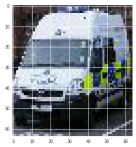

n02279972, Dis: 31.69


n03796401, Dis: 31.85


n03977966, Dis: 32.14


n04487081, 32.17


n03977966, 32.35


n03977966, 32.42

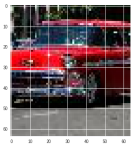

n02814533, 32.43

Figure 7: Bottom 10 images retrieved for Validation Set Image 2

it's relevance with all the other images in the same class.

$$r_i = \sum_{j=1, j\neq i, class_i = class_j}^{N} r_{i,j} \tag{2}$$

Hence, the probability of sampling a query image is proportional to it's total relevance score r_i.

Next, they sample a positive image from the same class as that of the query image with probability proportional to the relevance score r_{i,i^+} for query image p_i and positive image p_i^+ respectively.

$$P(p_i^+) = \frac{min(T_p, r_{i,i^+})}{Z_i} \tag{3}$$

Above is the probability of selecting a positive image, where T_p is the threshold, Z_i is the normalization and $r_{i.i^+}$ is the relevance score.

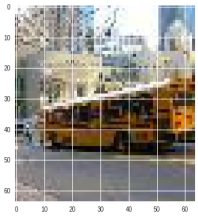
Next, to sample a negative image from out of class, uniform distribution is used, but to sample a negative image from the same class, the relevance score of $r_{i,i-}$ must be lower than the relevance of positive image i.e. r_{i,i^+}. Also, it is made sure that the difference between

Figure 8: Validation Set Image 3: n04146614

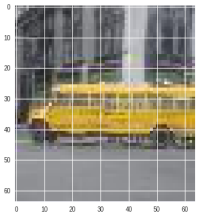
the relevance scores r_{i,i^+} and $r_{i,i-}$ is greater than a set threshold T_r.

$$r_{i,i^+} - r_{i,i-} \geq T_r \tag{4}$$

The decision of selecting an in-class or out of class negative sample depends on an out of class ratio parameter.

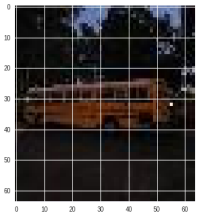


n04146614, Dis: 4.46

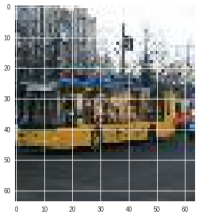


n04146614, Dis: 4.46

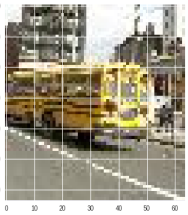
n04146614, Dis: 4.53



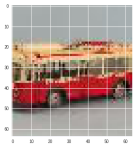
n04146614, Dis: 4.61



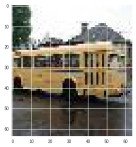
n04487081, Dis: 4.62



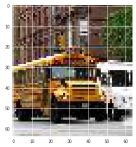
n04146614, Dis: 4.63



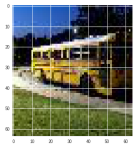
n04487081, 4.65



n04487081, 4.78

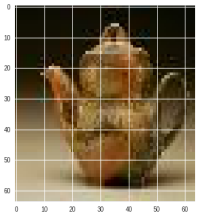


n04146614, 4.83

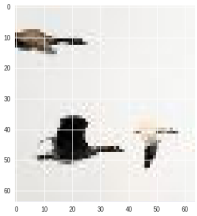


n04146614, 4.91

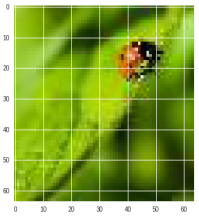
Figure 9: Top 10 images retrieved for Validation Set Image 3



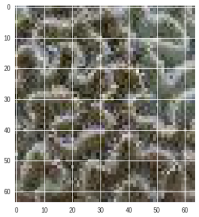
n04398044, Dis: 33.11



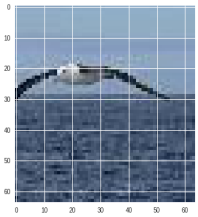
n01855672, Dis: 33.11



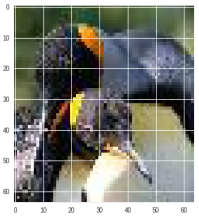
n02165456, Dis: 33.20



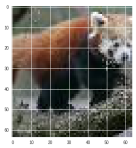
n01917289, Dis: 33.26



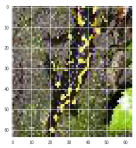
n02058221, Dis: 33.28



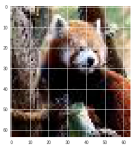
n02056570, Dis: 33.30



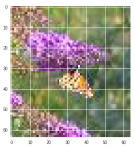
n02509815, 33.32



n01629819, 33.42



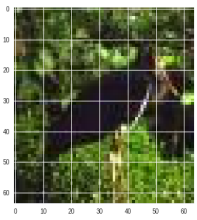
n02509815, 33.54



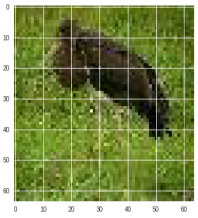
n02279972, 33.60

Figure 10: Bottom 10 images retrieved for Validation Set Image 3

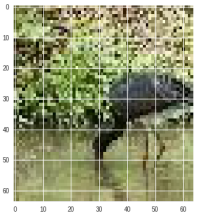
Figure 11: Validation Set Image 4: n02002724



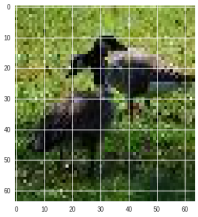
n02002724, Dis: 4.60



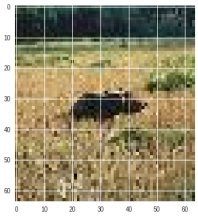
n02002724, Dis: 4.87



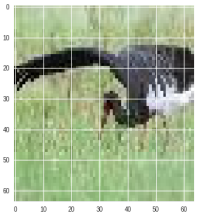
n02002724, Dis: 4.91



n01855672, Dis: 4.96

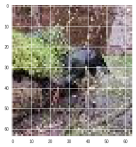


n02403003, Dis: 5.07

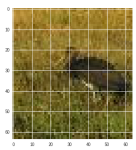


n02002724, Dis: 5.10

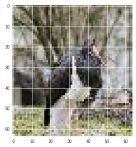
n02002724, 5.14



n02002724, 5.19

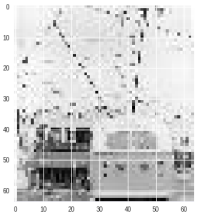


n02002724 5.2

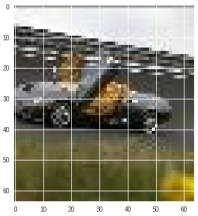


n02002724, 5.32

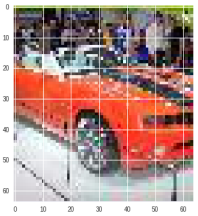
Figure 12: Top 10 images retrieved for Validation Set Image 4



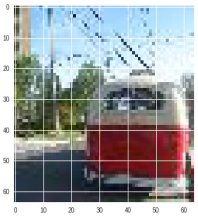
n04487081, Dis: 33.36



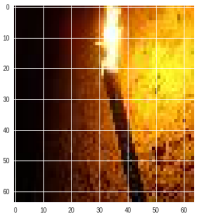
n04285008, Dis: 33.40



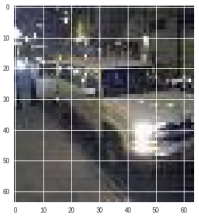
n03100240, Dis: 33.42



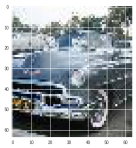
n04487081, Dis: 33.51



n04456115, Dis: 34.08

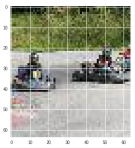


n03670208, Dis: 34.130

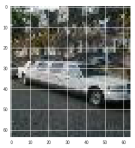


n03100240, 34.32

n03796401, 34.38



n03444034, 34.45

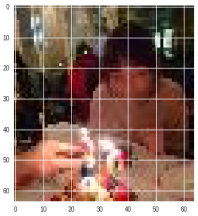


n03670208, 34.562

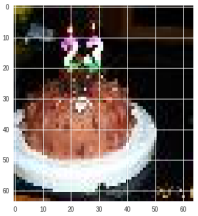
Figure 13: Bottom 10 images retrieved for Validation Set Image 4

Figure 14: Validation Set Image 5: n02948072

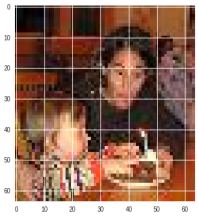
n02948072, Dis: 5.465



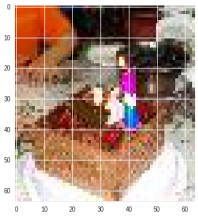
n02948072, Dis: 5.69



n02948072, Dis: 5.942

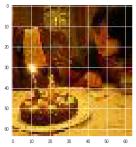


n02948072 Dis: 6.005

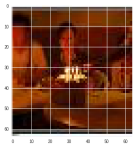


n02948072 Dis: 6.026

n02948072, Dis: 6.098

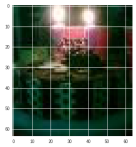


n02948072, 6.264



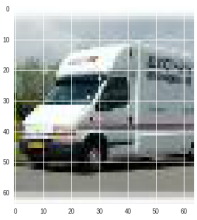
n02948072, 6.31

n02948072, 6.35

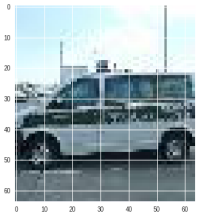


n02948072, 6.40

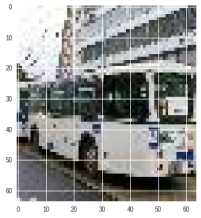
Figure 15: Top 10 images retrieved for Validation Set Image 5



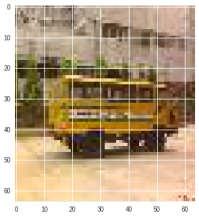
n03796401, Dis: 32.088



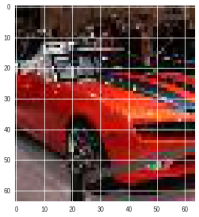
n03977966, Dis: 32.097



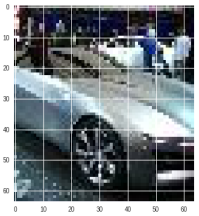
n04487081, Dis: 32.227



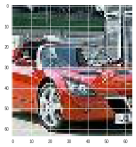
n04146614, Dis: 32.358



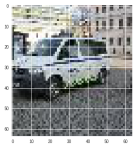
n03100240, Dis: 32.547



n04285008 Dis: 32.88

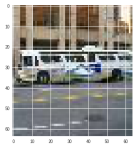


n04285008, 33.05



n03977966, 33.09

n02814533, 33.18



n04487081, 33.29

Figure 16: Bottom 10 images retrieved for Validation Set Image 5