University of Michigan
EECS 442: Computer Vision
Fall 2021. Instructor: Andrew Owens.

Problem Set 8: Representation Learning

Posted: Wednesday, Nov 10, 2021	**Due:** Wednesday, Nov 17, 2021

Please convert your Colab notebook to a PDF file and submit the PDF file to Gradescope. We have included the PDF conversion script at the end of the notebook. Nothing needs to be submitted to Canvas.

The starter code can be found at:
https://colab.research.google.com/drive/1C3B4Wf6Wqlp7FMr7fVL6ORaeQXcpeSjA?usp=sharing

We recommend editing and running your code in Google Colab, although you are welcome to use your local machine instead.

Problem 8.1 *Autoencoders* (5 pts)

We'll start by implementing a simple self-supervised learning method: an autoencoder. The autoencoder is composed of an encoder and a decoder. The encoder often compresses the original data with a funnel-like architecture, i.e., it throws away redundant information by reducing the layer sizes gradually. The final output size of the encoder is a *bottleneck* that is much smaller than the size of the original data. The decoder will use this limited amount of information to reconstruct the original data. If the reconstruction is successful, the encoder has arguably captured a useful, concise representation of the original data.

Such representations could help with downstream tasks such as object recognition, semantic segmentation, etc. Here, to test the usefulness of the representation, we'll train the encoders on the STL-10 dataset, which is designed to evaluate unsupervised learning algorithms. This dataset contains 100,000 unlabeled images, 5,000 labeled training images, and 8,000 labeled test images. To keep training time short, we'll use 10,000 unlabeled images to learn representations.

We'll then use the feature representation that we learned to train an object recognition model (a simple linear classifier) on the 5,000 labeled training images. If the learned representations are useful, we should obtain a performance improvement over only using the small, labeled training set.

1. We will build a small convolutional autoencoder and train it on the STL-10 dataset. The conv layers in the autoencoder all have kernel size = 4x4, stride = 2, padding = 1.

2. With the trained autoencoder, we freeze the parameter of the encoder and train a linear classifier on the autoencoder representations, i.e., the output of the encoder. You will compare

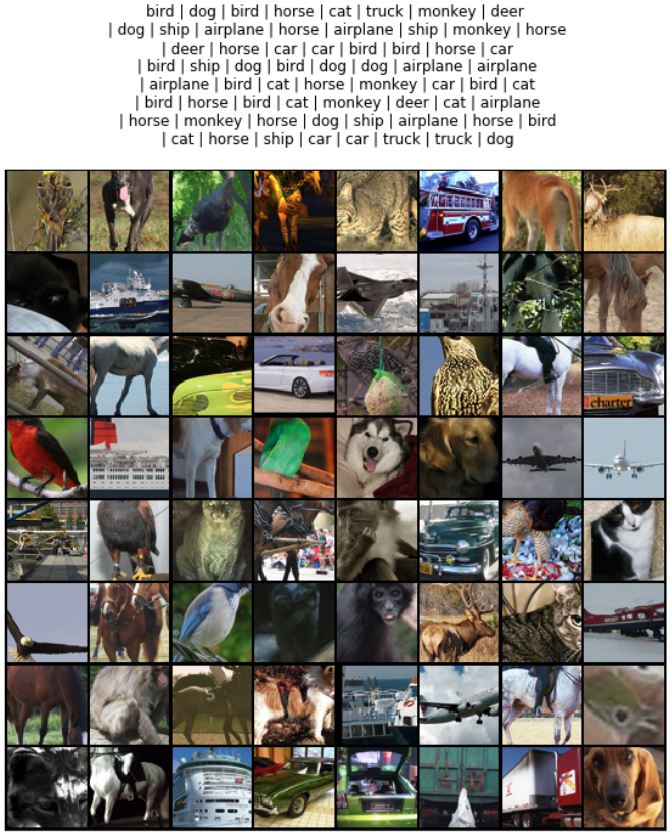

Figure 1: Sample images from STL-10 dataset.

the accuracy of the linear classifier with two other linear classifiers. One is trained together with the encoder and the other one is trained on top of a randomly initialized encoder. Confirm that the unsupervised pretraining improves the classification accuracy compared to the random baseline. Method I should achieve about 30% accuracy on the test set. Method II should achieve above 40% accuracy. Method III performs the worse among these three.

List of functions/classes to implement:

1. `class Encoder` (1 pt)
2. `class Decoder` (1 pt)
3. `def train_ae` (1 pt)
4. `def train_classfier` and set the supervised parameter in three methods (1 pt)
5. Report results in the Report results section at the end of the notebook (1 pt)

Problem 8.2 *Contrastive Multiview Coding* (5 pts)

We covered *contrastive learning* (CL) in lecture 15. CL is an approach of self-supervised learning [1, 2, 3] that avoids the need to explicitly generating images. Here, we'll implement a recent contrastive learning method, Contrastive Multiview Coding (CMC) [2]. We'll learn a

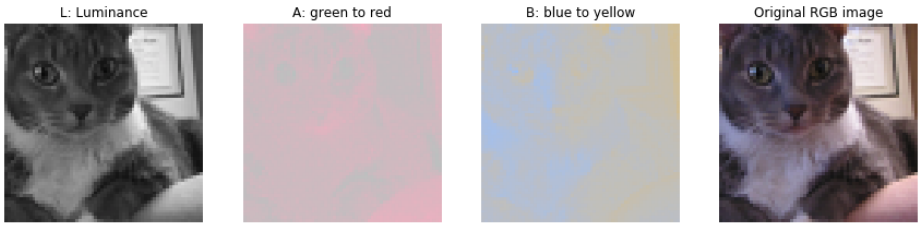

Figure 2: Lab channels.

vector representation for images: in this representation, two artificially corrupted versions of the same image should have a large dot product, while dot products of two different images should have a small dot product. In CMC, these corruptions are *views* of an image that contain complementary information. For example, in this problem set, our views will be luminance (i.e. grayscale intensity) and chromaticity (i.e. color) in the *Lab* color space. A good representation should create similar vectors for these two views (i.e. that have a large dot product), and they should therefore contain the information that is shared between the views. We'll minimize the loss:

$$\mathcal{L}_{\text{contrast}}^{V_1,V_2} = -\frac{1}{N}\sum_{i=1}^{N}\log\frac{h_\theta(v_1^i, v_2^1)}{\sum_{j=1}^{k+1} h_\theta(v_1^i, v_2^j)}, \tag{1}$$

where v_1 and v_2 are two different views of the data, k is the number of negative samples. The function h_θ measures the similarity between the representations of the two views, and is implemented using a neural network:

$$h_\theta(v_1, v_2) = \exp\left(\frac{f_{\theta_1}(v_1)\cdot f_{\theta_2}(v_2)}{\|f_{\theta_1}(v_1)\|\cdot\|f_{\theta_2}(v_2)\|}\cdot\frac{1}{\tau}\right), \tag{2}$$

and f_{θ_1} and f_{θ_2} are encoders for extracting representations from view 1 and view 2, respectively. The constant τ is the temperature hyperparameter for controlling the range of the numbers that are exponentiated.

We will minimize a symmetric objective function that sums $\mathcal{L}_{\text{contrast}}^{V_1,V_2}$ and $\mathcal{L}_{\text{contrast}}^{V_2,V_1}$, i.e.,

$$\mathcal{L}(V_1, V_2) = \mathcal{L}_{\text{contrast}}^{V_1,V_2} + \mathcal{L}_{\text{contrast}}^{V_2,V_1}. \tag{3}$$

By minimizing the above loss function, we learn representations from view 1 and view 2 such that the h_θ will give high scores for views of the same sample (positive pairs) while assigning low scores for views coming from different samples.

To represent our views, we'll use the luminance channel and chrominance channels of the Lab color space. Like the familiar RGB images, Lab images also contain 3 channels. The first channel contain brightness (luminance) information of the image while the other two channels contain color (chrominance) information. We visualize the three channels of a Lab image in Figure 2.

1. We will implement CMC loss functions and train two encoders. Finally, we will train a linear classifier on top of the CMC representation to classify the test data. There should be a a few percent higher performance improvement, compared to the autoencoder representations (Model II), for example, 48% v.s. 44%.(5 pts)

List of functions/classes to implement:

1. `class EncoderCMC` (1 pt)

   Hint: `torch.split` can help.

2. `class CMCScore` (1 pt)

   Hint: `torch.bmm` and `torch.index_select` can help.

3. `class SoftmaxLoss` (1 pt)

4. `train_cmc` (1 pt)

5. Report classifier accuracy at in the Report results section at the end of the notebook. (1 pt)

References

[1] Aaron van den Oord, Yazhe Li, and Oriol Vinyals. Representation learning with contrastive predictive coding. *arXiv preprint arXiv:1807.03748*, 2018.

[2] Yonglong Tian, Dilip Krishnan, and Phillip Isola. Contrastive multiview coding. *arXiv preprint arXiv:1906.05849*, 2019.

[3] Kaiming He, Haoqi Fan, Yuxin Wu, Saining Xie, and Ross Girshick. Momentum contrast for unsupervised visual representation learning. *arXiv preprint arXiv:1911.05722*, 2019.