EE599 Assignment 4: Deep Learning for Classification in Computer Vision

March 10, 2020

The goal of this assignment is to apply deep learning to computer vision. Particularly, you'll work with the classification problem on a fashion compatibility dataset called Polyvore. Your goals will be to set up your category classifier and fashion compatibility classifier. Turn in your code and report as described in Section 5.

The starter code for this project can be found at https://github.com/davidsonic/EE599-CV-Project but feel free to explore different hyper-parameters and model structures. Follow the instruction in the readme file to setup the codebase.

1 Dataset Description

Polyvore Outfits [1] is a real-world dataset created based on users' preferences of outfit configurations on an online website named *polyvore.com*: items within the outfits that receive high-ratings are considered compatible and vice versa. It contains a total of 365,054 items and 68,306 outfits. The maximum number of items per outfit is 19. A visualization of an outfit is shown in Figure 1.

Figure 1: A visualization of a partial outfit in the dataset. The number at the bottom of each image is the ID of this item.

2 Category Classification

- The starter code provides the following files with blanks in them and can be read in this order. First, you need to set your dataset location in utils.py (Config['root_path']).

  1. train_category.pytrain_compat.py: training scripts
  2. model.py: CNN classification models
  3. data.py: dataset preparation
  4. utils.py: utility functions and config

- Training takes place in train_model function of train_*.py. In each iteration, the model takes in batches of data provided by the dataloader (data.py). Record your training acc progress here, which will be used for plotting the learning curve.

- You're expected to do finetuning and training from scratch.

  1. Finetune a model pretrained on ImageNet (e.g, ResNet50). Frameworks nowadays provide easy access to those, refer to documentations online.
  2. Construct a model of your own and start training from scratch.
  3. Compare these two models and record the results. What is the advantage of using a finetuned model? What's the difference between the learning rates when you apply these two learning strategies (i.e finetuning vs from scratch)?

  Note: images are located in images folder, each image is named by its id. Information for the item are stored in train_category.json.

- Modify data.py to create data pairs (image, category label). Normalization is defined in get_data_transforms function.

- Split no less than 10% data for validating your final model. The test set is test_category_hw.txt.

- **Tips:**

  1. Over-fitting is expected. Play with model structure and hyper-parameter or regularization to reduce over-fitting. You can design any model structure you like.
  2. To speed up the training speed, you can set "use_cuda" flag to true and increase the batch_size defined in utils.py.
  3. You can restrict the size of the dataset for quick debugging. Set debug=True in utils.py. You do not necessarily need to use 20 epochs and the entire training set.
  4. It may take more than several epochs before the performance plateaus, depending on the network structure you use and the learning rate.

3 Pairwise Compatibility Prediction

The task is to predict the compatibility of an outfit (Figure 2). It's essentially a binary classification problem (compatible or incompatible), however, the difficulty of the task lies in the input–classify based on a set rather than a single item as you did in the last section. One idea to deal with set classification is to decompose it into pairwise predictions (you're encouraged to propose different ideas for the bonus section). Therefore, you'll first train a pairwise compatibility classifier.

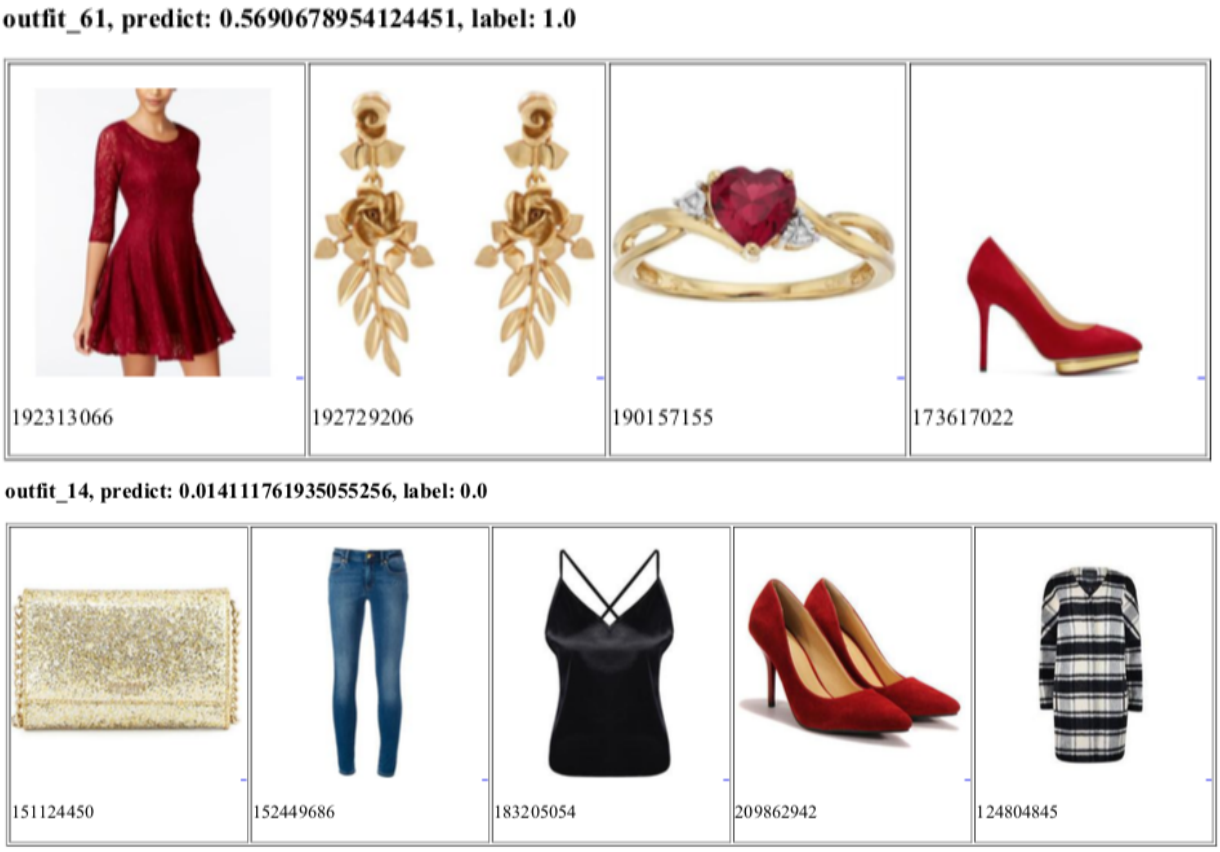

Figure 2: Examples of a compatible item and an incompatible item.

- Modify data.py to create a new dataloader that takes in a pair of image inputs (compatible pair and incompatible pair). For example, let's assume any pair of items in a compatible outfit are considered compatible whereas an incompatible outfit provides negative pairs.
- Modify model.py to create a new model that takes in a pair of inputs and outputs a compatbility probability for this pair.
- Split no less than 10% your data for validation. The test set is test_pairwise_compat_hw.txt.

- **Bonus**: Make outfit compatibility prediction based on pairwise predictions (i.e, average over n(n-1)/2 pairwise scores and then set a threshold for outfit compatibility). The test set is compatibility_test_hw.txt.
- **Tips:**
    1. Outfit descriptions are located in compatibility_*.txt. Each line shows the compatibility of the outfit and its items. (e.g, 1 210750761_1 210750761_2 210750761_3: a compatible outfit id, whose outfit id is 210750761. It has three items, indexing from 1 to 3).
    2. Item ids and descriptions are located in polyvore_item_metadata.json. Their corresponding images are also indicated by item ids.
    3. To associate items in an outfit with their item id, you need to parse train.json/val.json.

4 Extra Bonus

Considering this is a real-world data and that fashion compatibility prediction is an open problem. You're encouraged to refine the performance by adding various tricks.

- Perform learning rate scheduling
- Perform data augmentation to increase robustness
- Perform hard-negative mining for pairwise compatibility prediction
- Learn a permutation invariant feature for the entire set for compatibility prediction

5 Turning In

- **Submitting the PDF** Make a PDF report containing: answers for what the question marks in this instruction and a table of results for categorization accuracy and pairwise compatibility accuracy (Extra Bonus: outfit compatibility accuracy).
- **Submitting the code** The folder with all python files.
    1. Plot for the model structure you designed (category-model.png/compatibility-model.png).
    2. Learning curves during training of category classifier and compatibility classifier (category.png/compatibility.png).
    3. A category.txt file containing two columns: Items ID and predicted category.
    4. A pair_compatibility.txt with three columns: item1, item2 and prediction score.
    5. Extra Bonus: A outfit_compatibility.txt file containing two columns: outfit ID for the test set and predicted compatibility.

References

[1] Mariya I Vasileva, Bryan A Plummer, Krishna Dusad, Shreya Rajpal, Ranjitha Kumar, and David Forsyth. Learning type-aware embeddings for fashion compatibility. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 390–405, 2018.