Assigned: 22 October 2020

Deep Learning – Homework #4

EE 599: Fall 2020

Due: Monday, 03 November 2020 at 20:00. Submission instructions will follow separately on Canvas.

0 Overview

In this assignment you will apply deep-learning techniques to computer vision. You will devise a solution to a fashion compatibility problem based on the Polyvore dataset. Your task is to create a category and fashion compatibility classifier.

Find (minimal) starter code for this project at https://github.com/franzke-usc-fa20/ee599-fa20-hw4-starter. We expect you to explore and tune hyper-parameters and model architecture beyond the starter code. Follow the instruction in the readme file to setup the codebase.

1 Dataset Description

The *Polyvore Outfits* [1] dataset is *real-world* data created from users' outfit preferences at polyvore.com. Consider paired items from outfits that receive high-ratings as *compatible*. The Polyvore dataset contains 68,306 outfits created from 365,054 items. There is a maximum 19 items per outfit. Figure 1 shows an example outfit.

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 repository includes several files. Some are almost blank but give a structure to build from.

  First: make sure that you set the dataset location in `utils.py` (`Config['root_path']`).

  1. `train_category.py`: training script

2. `model.py`: CNN classification models
3. `data.py`: dataset handlers
4. `utils.py`: utility functions and configuration

- The `train_model` function in `train_*.py` handles model training. The driver applies a batch of data to the model from the dataloader (`data.py`) during each iteration. You should modify the code to record your training accuracy so you can review learning curves.
- Fine-tune your model and train "from scratch":

  1. Finetune a pretrained ImageNet model (e.g, ResNet50). The PyTorch model-zoo allows ready access to many standard CNN architectures.
  2. Construct your own model and train from scratch.
  3. Compare the two models (pretrained vs. custom) and comment on the results. What is the advantage of fine-tuning a pretrained model vs. creating a custom network? Did you require learning rates for each model?

  Note: the images folder contains all the data images coded by its id. You can find more information about each image in `polyvore_item_metadata.json`.
- Modify `data.py`. Create supervised training pairs (image, category label). The `get_data_transforms` function applies input normalization.
- Split **no less than 10% data** for testing your final model. Generate a category.txt that lists item ids from your test set and the predicted category vs ground-truth category.
- **Tips:**

  1. Expect over-fitting. Tweak the model structure, hyper-parameters, and regularization to reduce over-fitting. You can design any custom model structures you like.
  2. Set `to(device)` flag (`device=cuda:0`) and increase the `batch_size` defined in `utils.py` to speed up training.
  3. To debug try reducing the data-set size. Set `debug=True` in `utils.py`. You can also reduce the number of epochs and set a random `seed` to aid debugging.
  4. It will take many epochs to reach a performance plateau. But this will depend on the network structure and the learning rate(s). Explore whether training further increases generalization or tends toward over-fitting.

3 Pairwise Compatibility Prediction

You are tasked to predict outfit compatibility (Figure 2). Interpret this the compatibility-prediction as a binary classification problem (*i.e.* compatible or incompatible). The challenge stems from the `input` → `classification` based on a set rather than a single item as in the previous section. Think of creative ways to deal with the set classification. The next section steps you through aggregate classification based on pairwise predictions – but we encourage you to experiment with new ideas to improve your classification performance.

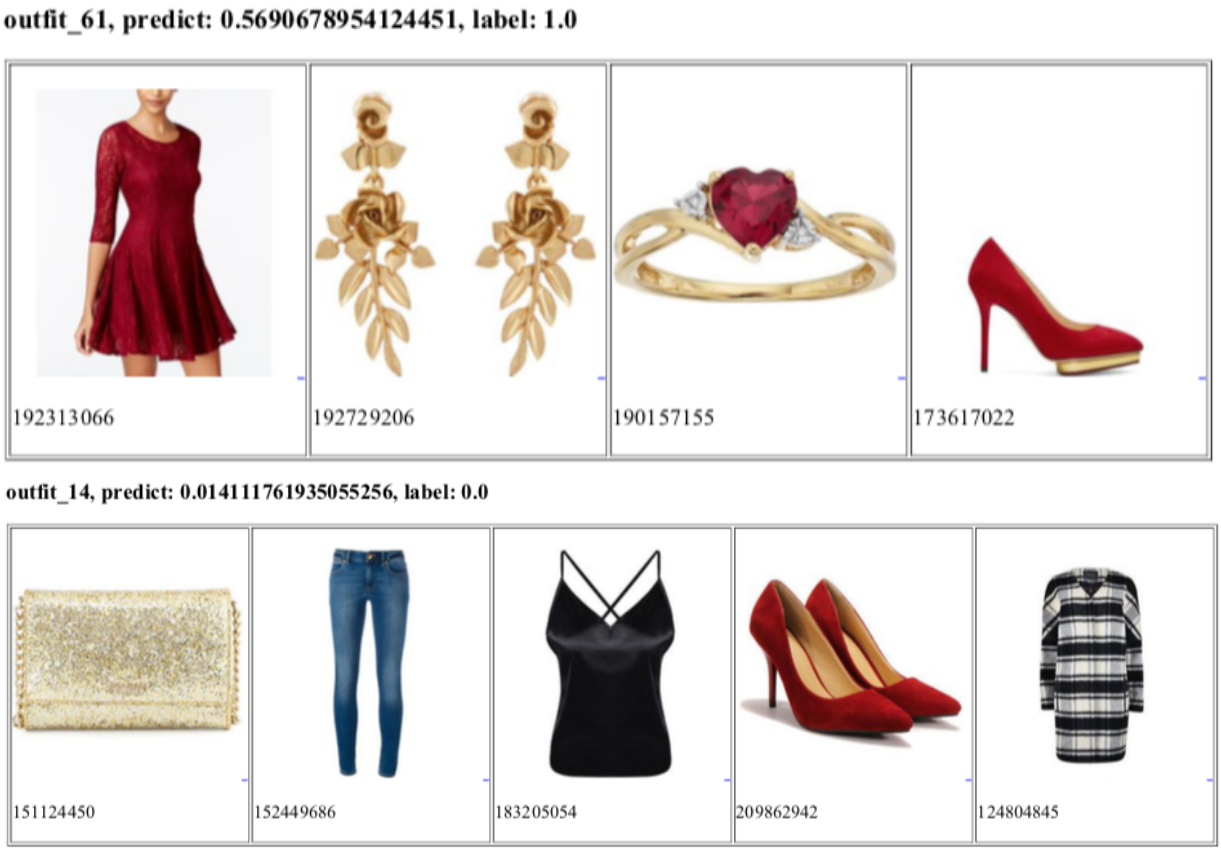

Figure 2: Examples of a compatible item and an incompatible item.

- Create a new dataloader by modifying `data.py`. Return pairs of image inputs (*i.e.* compatible pairs vs incompatible pairs). For example: assume any pair of items in a compatible outfit are "compatible" whereas incompatible outfits provide negative pairs.
- Modify `model.py` and create a model that takes pairs of inputs and outputs a compatibility probability for the pair.
- Split **no less than 10%** your data for testing. Generate a `pair-compatibility.txt` file that lists `item1-id`, `item2-id`, `predicted compatibility score`, and `ground-truth compatibility`.
- **Bonus**: Make outfit compatibility prediction based on pairwise predictions (i.e, average over $n(n-1)/2$ pairwise scores and choose an outfit-compatibility threshold).
- **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. `polyvore_item_metadata.json` contains item ids and descriptions. The corresponding images referenced by item id.
3. Associate items in an outfit with their item id by referencing `train.json` (`val.json`, `test.json`).

4 Challenge/Bonus Tasks

This is a real-world dataset and predicting fashion compatibility is an open problem. You are encouraged to experiment with models and parameters to enhance performance. Don't feel above "tricks" to improve generalization. Focus on performance of an adequately partitioned test-set and tuning such as:

- learning rate scheduling
- data augmentation to increase robustness
- hard-negative mining for pairwise compatibility prediction
- permutation invariant feature detection for the dataset to enhance for compatibility prediction

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.