Learning Representations of Sets through Optimized Permutations

https://github.com/Cyanogenoid/perm-optim

Yan Zhang, Jonathan Hare, Adam Prügel-Bennett

University of Southampton

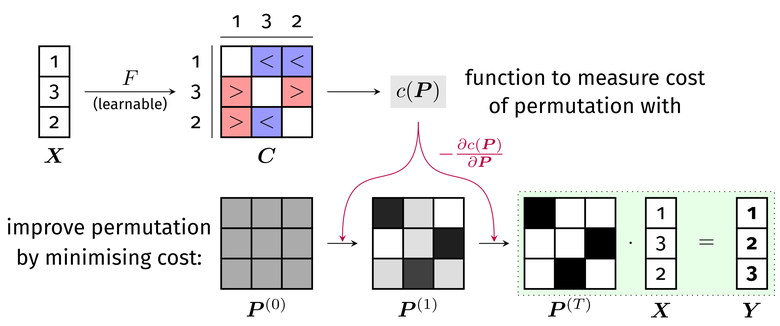
Introduction

Summary

To obtain a set representation, learn how to permute a set into an ordered representation, then encode permuted set with an RNN or CNN.

- **Sets** are *unordered* collections of things:
  - the set of objects in an image,
  - the set of points in a point cloud,
  - the set of papers submitted to a conference,
  - the set of nodes in a graph.
- How do we do Machine Learning with sets as input?
- **This paper**: learn *latent permutation* to permute the set into a canonical sequence, then use a traditional sequential model (e.g. LSTMs) to obtain a set representation.
  - doesn't suffer from bottleneck of sum or max pooling,
  - end-to-end trainable,
  - adapts to what the sequential model "likes",
  - works for permutations on lattices, such as image tiles on grids.

Permutation-Optimisation module (PO)



- **Cost function** $c(\boldsymbol{P})$ measures "quality" of permutation based on pairwise ordering costs $\boldsymbol{C}$.
  - Cost of placing element a anywhere before b is C_{ab}.
  - Cost of placing element a anywhere after b is $-C_{ab} = C_{ba}$.
  - Weighted by the current soft assignment in $\boldsymbol{P}$.
- **Gradient descent** of $\boldsymbol{P}$ for T steps to minimise the cost.
- **Comparison function** $F(a, b)$ produces entries of $\boldsymbol{C}$.
  - Negative when element a should be before b in the sequence, positive when a should be after b.
  - Property enforced by parametrising $F(a, b) = f(a, b) - f(b, a)$.
- Soft permutations $\boldsymbol{P}$ (entries can be between 0 and 1) to allow differentiation through the optimisation of $\boldsymbol{P}$.
- Sinkhorn normalisation of $\boldsymbol{P}$ so that rows and columns of P always sum to 1.

Image mosaics

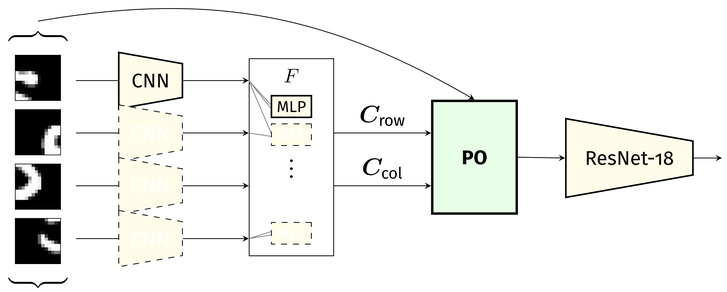


Figure 1: Architecture for image mosaics. Classification from a set of image tiles instead of the original image. This uses comparisons left-to-right (C_{row}) and top-to-bottom (C_{col}).

The network is not told what the correct image should look like; it learns this implicitly by learning to classify.

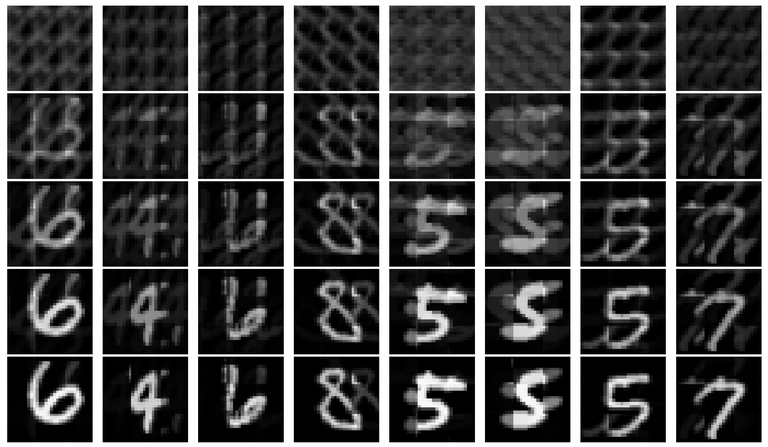


Figure 2: Reconstructions on MNIST 3x3 as they are being optimised.

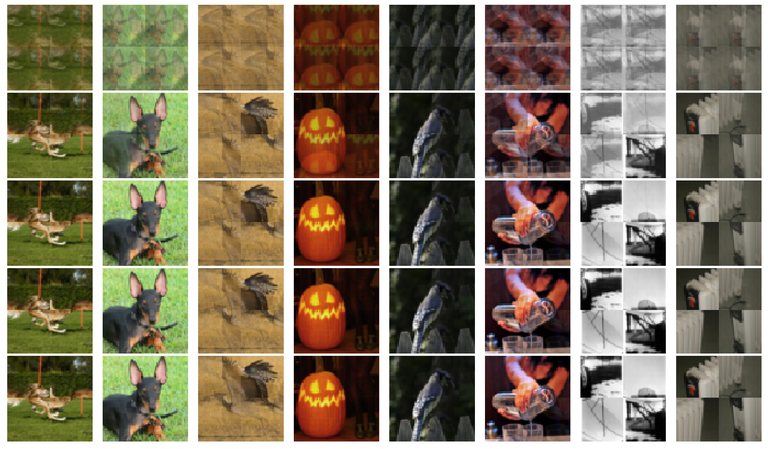


Figure 3: Reconstructions on ImageNet 2x2 as they are being optimised.

Analysis

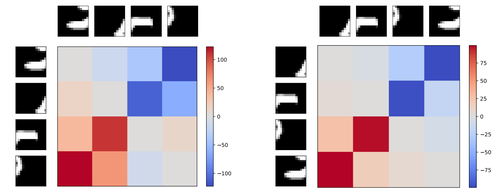


Figure 4: Comparison of tiles left-to-right (left) and top-to-bottom (right). Blue entries denote that the tile to the left of this entry should be on the left side (left figure) or above (right figure) of the tile above the entry. Red entries denote the opposite.

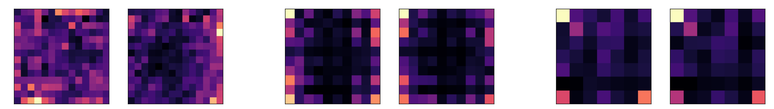


Figure 5: Sensitivity of comparison to pixel locations within a tile on MNIST 2x2 (left), 3x3 (middle), 4x4 (right), for left-to-right and top-to-bottom comparisons.

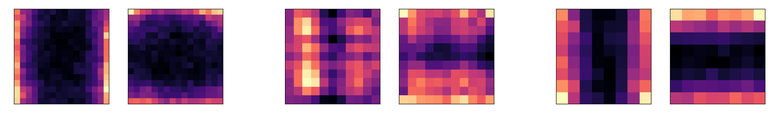


Figure 6: Sensitivity of comparison to pixel locations within a tile on CIFAR10 2x2 (left), 3x3 (middle), 4x4 (right), for left-to-right and top-to-bottom comparisons.

Visual Question Answering

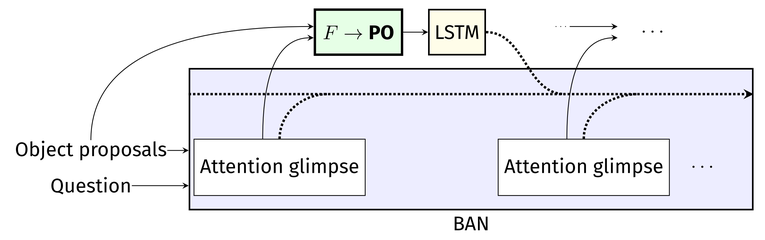


Figure 7: Architecture for VQA. Simple modification of state-of-the-art BAN model (Kim et al., NeurIPS 2018). Permute the set of object proposals, then encode resulting sequence with an LSTM. The encoding is fed back into the BAN model.

Table 1: State-of-the-art results on VQA v2. Classification accuracy on validation set over 10 runs.

Model	**Overall**	Yes/No	Number	Other
BAN	65.96	83.34	49.24	57.17
BAN + LSTM	66.06	83.29	49.64	57.30
BAN + LSTM + PO	**66.33**	**83.50**	**50.42**	**57.48**