AI3607 Task4 Report

Weijiude 521030910418

June 1, 2023

1 Introduction

We imitate the model architecture from DeepPermNet to solve the puzzle permutation problem on our artifact benchmark based on CIFAR-10. We evaluate the performance of model with sinkhorn normalization and another model in which sinkhorn normalization is replaced by independent sigmoid normalization. We found that sinkhorn normalization can effectively increase the accuracy of permutation based on independent position prediction. Then we attempt to find the relationship between model performance and padding size. We initially propose that padding could protect the margin information of images but the experiment shows no evidence to support our proposal. A synchronized matching between padding size and kernel size can best the capacity of extracting features in CNN. We also evaluate the performance of model on puzzles illustrating various items and puzzles with different sizes. The result shows that our model is capable of complete simple puzzles with big enough area that illustrates items with clear contrast ratio and geometrical shape.

2 Puzzle Permutation

2.1 Task

We divide an image into n parts, then shuffle the n parts corresponding with a random permutation of their original order. We need to fit a model that takes the shuffled parts as input and able to return the original order of each part.

Formally, we define X is a sequence of n image pieces in its original order. In addition, consider an artificially permuted version $\hat{X}$ where the image pieces in X are permuted in a random order corresponding with a permutation matrix $P \in \{0,1\}^{n\times n}$. Namely $\hat{X} = PX$. P is by definition a double stochastic matrix, a square matrix constituted of non-negative reals, where all rows and all columns sum to 1.

Given a training set $\mathcal{S} = \{(\hat{X}, P) \mid \hat{X} \in \mathcal{X}, P \in \mathcal{P}^n\}$ where S is the permutations of image pieces and $\mathcal{P}^n$ represents n-order double stochastic matrices. We would like to fit a parameterized function $f_\theta : \mathcal{X} \to \mathcal{P}^n$ which takes a permutation of image pieces as input and returns the probability that an image part is at a certain position in its original permutation. Therefore the puzzle permutation problem can be described as

$$\arg\min_{\theta} \sum_{(\hat{X},P)\in\mathcal{S}} \mathcal{L}(P, f_\theta(\hat{X}))$$

where $\mathcal{L}$ is a loss function.

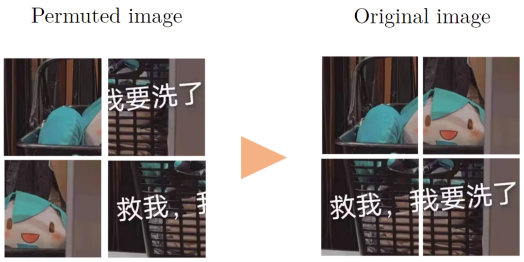

Figure 1: Sample of 2×2 puzzle permutation

2.2 Model Architecture

The very first step of puzzling is to capture the features of each image's margin. We imitate the AlexNet to use a convolutional neural network to capture the features. However, trivial CNN may not emphasize or even neglect the margin of images, which is important in puzzling. Therefore we propose to pad the images before feed them into CNN in order to protect their marginal features. Each image part is taken as input respsectively while the CNNs share same weights.

The CNNs would output n features corresponding to each image pieces. We concatenate the features and use several fully-connected layers to map the concatenated feature on a $n \times n$ matrix. Here each row of the matrix have already described the probability that an image part is at a certain position. However, the greatest item of multiple rows may be located in the same line. That means sevaral image parts may share a same position in original permutation, which is nonsense in puzzling.

To distribute each image pieces to different positions, we cast sinkhorn function on the matrix given by MLP to transform it into a double stochastic matrix. The property of double stochastic matrix promises that the image pieces are distributed to unique positions.

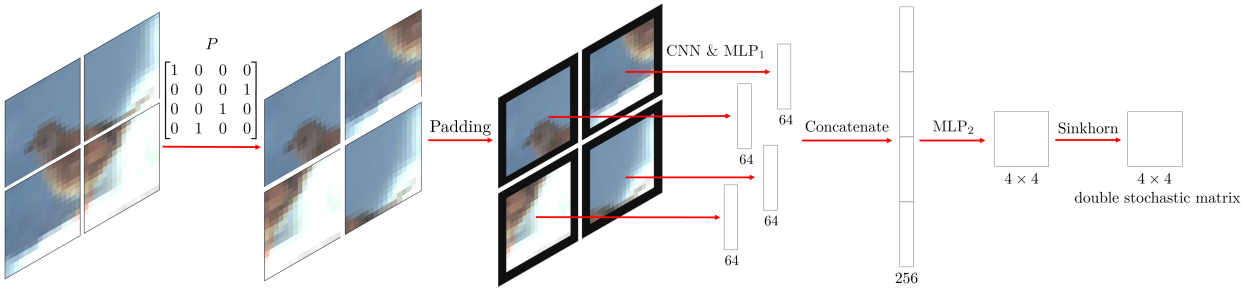

Figure 2: Implementation of model architecture

2.3 Implementation

We divide an image into 4 parts horizontally and vertically. Then we shuffle the image pieces with respect to a randomly permutation.

Our model pads each image part with 2 pixels firstly. Then each image part is taken into a CNN and subsequent MLP which constitutes of following layers sequentially:

1. convolutional layer with kernel size 3 to raise channel number from 3 to 16
2. maxpooling layer with kernel size 2 and stride 2
3. convolutional layer with kernel size 3 to raise channel number from 16 to 32 (flattened later)
4. fully-connected layer with input feature 3200 and output feature 128
5. Relu as activate function
6. fully-connected layer with input feature 128 and output feature 64

Then the four features with length 64 are concatenated into one vector with length 256. After that an MLP takes it as input and outputs a vector with length 16. We flatten the vector into a 4×4 matrix. The matrix is transformed into a double stochastic matrix through a sinkhorn function.

We use cross-entrophy loss as loss function given double stochastic matrix a two-dimensional probability distribution.

3 Experiment

3.1 Puzzle Permutation Prediction

In this section we evaluate our method on the puzzle permutation prediction task. Also we compare our method with a simpler method where the sinkhorn is replaced by a four-way sigmoid function, which treats each image part as a multi-classfication task independently.

We can observe from Table 1 that the model with sinkhorn normalization outperformances the one with four-way independent sigmoid function. The accuracy of predicting the permutation of image pieces is nearly half the accuracy of predicting the positions of image pieces independently with the four-way sigmoid function, which indicates that this method is capable of learning the normalization by itself to some extent.

Method	Ind	Perm	Perm / Ind
Sinkhorn Norm	**0.819**	**0.734**	89.6%
4-way sigmoid	0.804	0.632	79.0%

Table 1: Evaluation and comparation of Sinkhorn normalization and four-way sigmoid normalization. Ind stands for the accuracy of predicting each image part's position independently. Perm stands for the accuracy of predicting the permutation of all image pieces.

On the other hand, the model with sinkhorn normalization reaches a high permutation accuracy based on its high accuracy of predicting image pieces independently. The percentage of permutation prediction accuracy in independent position prediction accuracy is significantly increased with sinkhorn normalization, which indicates that it is useful in fitting a multi-dimensional probability distribution.

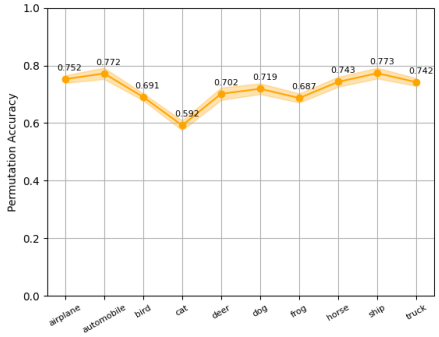

Figure 3: Permutation performance of various image labels. Evaluated on bench with method of kernel size 3, sinkhorn normalization and various padding sizes.

We wonder the performance of our puzzle permutation method on different kind of images. We evaluate the

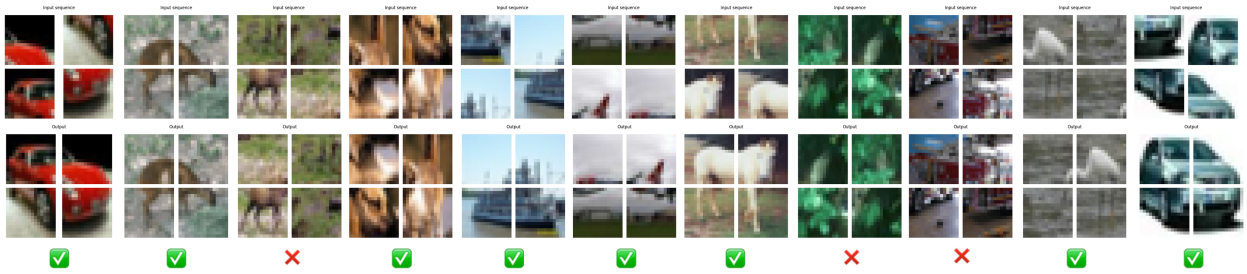

Figure 4: Sample of puzzle pertation prediction with sinkhorn normalization, padding size 2 and kernel size 3. The images above are input. The images below are responded output through our model.

puzzle permutation accuracy over all CIFAR-10 labels with same kernel size as 3, padding as 2 and sinkhorn normalization. The result is shown in Figure 3.

We can conclude from Figure 3 that our permutation method outperformances on images illustrating airplane, automobile, ship and truck. In addition to the observation of some sample of permutation prediction (see Figure 4), the common property of items in images that our method outperformances is clear boundary, geometrical shape, high contrast ratio and capacity of distinguishing itself from background.

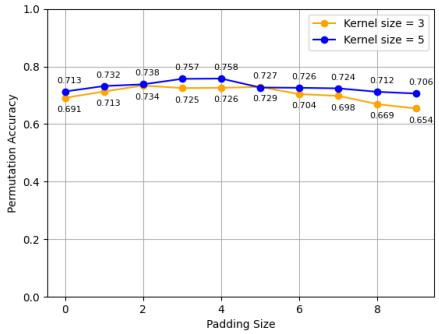

Figure 5: Evaluation and comparation of permutation accuracy based on various padding size.

3.2 Padding

We are curious about how padding can perserve the marginal information and what padding size will outperformance. The convolutional layers and pooling layers may obscure the marginal features with central features and hence weaken the weight of margin in overall information. Based on such idea, we proposed to pad the image pieces before feeding them into CNN. Meanwhile, the capacity of extracting features also depends on the kernel size of convolutional layers and the pooling size of pooling layers.

In this section, we evaluate the performance of models trained with various padding size under situation where kernel size is either 3 or 5.

We can indicate from Figure 5 that the permutation accuracy increases with a period of padding size. This phenomenon is most obvious in small padding size and the increase of permutation accuracy is slighter as padding size is growing. The model trained with kernel size 3 performances best when padding size is 3 while the model trained with kernel size 5 performance best when padding size is 5.

We take the results where kernel size is 3 as instance. We pick an image and print its feature given by a specific convolutional layer in our models with same hyperparameters except that padding size varies from 0 to 5. We can observe from Figure 6 that the inner features of a certain channel are similar among

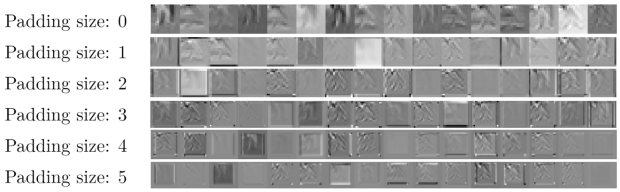

Figure 6: Output of the convolutional layer of the same image under various padding size. Each row stands for 16 channels in the output given by model trained with certain padding size.

different padding sizes. An outer square fence is getting more obvious and thicker as the padding size increases, which reflects the result of convolutional layer on padded zeros. Such phenomenon is not obvious before padding size is not greater than 2, which may explain that the permutation accuracy reaches its peak at padding size 3. Such zero fence reduces the ratio of informative pixels among all pixels. The increase of noise-signal-ratio has weakened the training effeciency and hence reduces the prediction accuracy.

In conclusion, we have found no strong evidence to back up our previous proposal that padding could protect the margin information of images. With certain kernel size, a suitable padding size can maximize the training effeciency and prediction accuracy, which is an integer close to kernel size. A huge padding size

can enlarge the noise-signal-ratio and hence weaken the learning capacity of models.

3.3 Generalization

We also wonder generalization capacity of our model. We enlarge our test bench based on CIFAR-10 dataset to puzzles with size 3×3. We import some functions from *cv2* to enlarge the size of images so that each iamge part in both 3×3 puzzle and 4×4 puzzle has side len 16, same with the side len of image pieces in 2×2 puzzle.

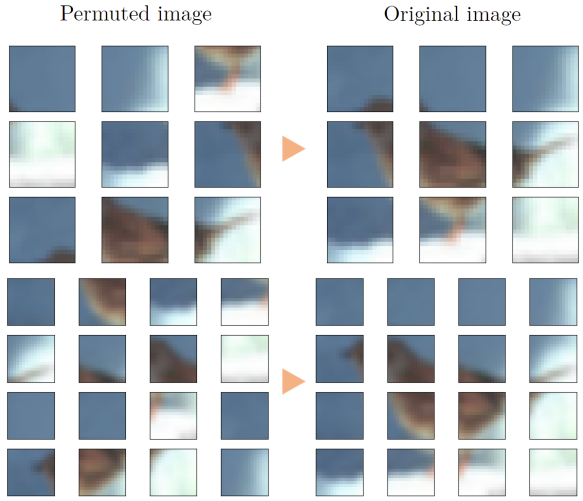

Figure 7: Sample of generalized evaluation of 3×3 puzzle (above) and 4×4 puzzle (below). Our model receives a random permutation of image pieces and make a prediction of its correct permutation.

We remaster our model architecture implementation for a $n \times n$ puzzle($n = 2, 3, 4$). Some essential changes are listed below.

1. A $n \times n$ ground truth 01 permutation matrix.
2. $n \times n$ parallel weight-shared CNNs and subsequent MLPs, each of which has implementation same as the implementation in 2×2 puzzle.
3. Concatenate $n \times n$ features with length 64 into one vector with length $64 \times n \times n$. Another MLP takes this vector as input and outputs a vector with length $n \times n$.
4. We flatten the vector into $n \times n$ matrix and cast sinkhorn function to transform it into a double stochastic matrix as prediction.
5. We set both padding size and kernel size as 3 given that our model performs best in 2×2 puzzle with such hyperparameters.

Figure 8 depicts the permutation accuracy evaluated on 2×2, 3×3 and 4×4 puzzles. Despite our previous good performance on 2×2 puzzles, our model performs worse as the piece number of puzzles increases. Our model has still learned some weak features from puzzles with larger scale, given that the permutation accuracy is higher than random guess.

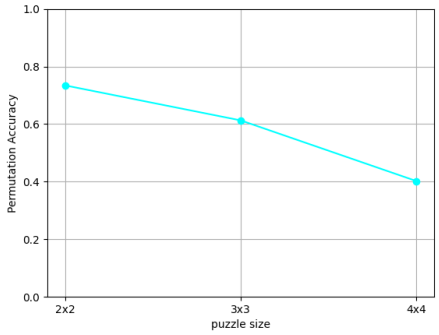

Figure 8: Evaluation of generalization capacity of our model, with kernel size 3, padding size 2 and sinkhorn normalization.

In conclusion, our model is capable of learning permutation features from simple puzzle tasks where the images have clearly bounded, geometrically shaped items with larger area and less number of seperated parts.

4 Conclusion and Discussion

In this project, we proposed to solve puzzle permutation task in deep learning way. We use several parallel weight-shared CNNs to extract the features of images pieces, then use an MLP to predict the original permutation where input is concatenated CNN outputs. To strengthen the capacity of non-duplicate prediction, we would like to cast sinkhorn function on the prediction matrix to transform it into a DSM, a two-dimensional probability distribution.

Due to the constant knowledge of puzzle games, we initially put emphasis on margin information of image pieces to enhance our model's performance on puzzle permutation. But we found no strong evidence in later experiment that supports enough padding could protect margin information in image pieces and even raise the model performance. The selection of padding size is dependent with kernel size. An oversized padding size could raise the noise-signal-ratio in features given by CNN and hence influence the effeciency of subsequent MLP. A CNN with padding size that is too small would neglect some information in image to some extent and also decrease the model performance. A padding size close to kernel size is usually better in our model.

Our model is still weak in solving complicated puzzle task. The comparation of permutation accuracy

among various item in target images shows that our model performs better if the item has clear boundary, geometrical shape and high contrast against background, which is usually artifact such as vehicle, airplane or ship. Images with natrual animal item, especially when the item looks vague in its background, is hard to be correctly reserved by our model. The result of generalization also demonstrates that our model lacks ability to solve puzzle with a large number of pieces, which raises the puzzle complexity in another way.

We construct a simplified DeepPermNet in this project and its performance slightly falls behind the DeepPermNet baseline. We discussed with some students who use a complete DeepPermNet in this project. DeepPermNet performs better than our simplified model in almost all item that images depict. It also shows no significant drop when the number of pieces in puzzle grows, which means DeepPermNet is robust in generalized puzzle tasks. Meanwhile some students who use even more complicated model based on DeepPermNet could still not keep up with the performance of DeepPermNet in baseline 2×2 puzzle evaluation. A model may not perform better if it is more complicated. Some gradient might be too small in backward chain and some parameters might be fogotten in a deep model.

5 Acknowledgement

In this final project, I would like to appreciate *Prof. Junchi Yan* and *Prof. Wei Shen* for providing us deep learning theories, basic computer vision theories and experience of scientific reasearch in this semester.

I would also like to appreciate *T.A. Chang Liu* for planning the project and answering our questions in time with high quality.