AI3607 Deep Learning: Course Project

Xiangyuan Xue (521030910387)
School of Electronic Information and Electrical Engineering

I. Task Description

In this project, we will research the jigsaw puzzle problem on CIFAR10 dataset. To be specific, given a complete natural image, we split it into multiple sub-images and shuffle them. Then we need to design an end-to-end neural network to rearrange the shuffled sub-images and merge them into the original image. A 2×2 sample is shown as follows.

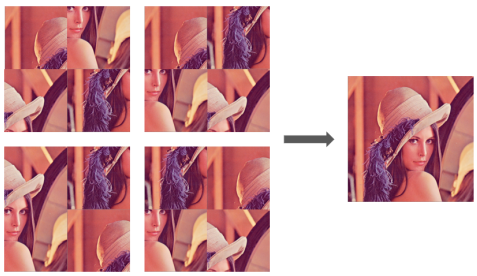

Fig. 1. jigsaw puzzle

To formalize, the sub-images are taken as the inputs and the permutations are taken as the labels. The neural network is expected to output a permutation matrix $\boldsymbol{P} \in \{0,1\}^{n \times n}$ where $n = k^2$ represents the number of sub-images and $\boldsymbol{P}_{ij} = 1$ indicates that the i-th sub-image is placed at the j-th position. There should be only one 1 in each row and each column of $\boldsymbol{P}$, which can be expressed as

$$\boldsymbol{P} \cdot \mathbf{1}^n = \mathbf{1}^n \tag{1}$$

$$\boldsymbol{P}^T \cdot \mathbf{1}^n = \mathbf{1}^n \tag{2}$$

which will be discussed in detail later in the report.

II. Dataset Construction

Download CIFAR10 dataset from the source. Read the images and normalize them to $[0, 1]$, where the raw labels are deprecated. The scale of the jigsaw puzzle can be 2×2, 3×3 or even larger. Considering the complexity of permutation, the size of the dataset increases exponentially with the scale of the puzzle, which is shown in the following table.

TABLE I
DATASET SIZE

Scale	#Class	Set	#Size
2×2	4!	Train	$50000 \times 4!$
		Test	$10000 \times 4!$
3×3	9!	Train	$50000 \times 9!$
		Test	$10000 \times 9!$
4×4	16!	Train	$50000 \times 16!$
		Test	$10000 \times 16!$

When an image is loaded, we split it into n sub-images and generate a random permutation of $1, 2, \ldots, n$. Then the sub-images are shuffled accordingly and the permutations are taken as the labels. Note that only one version of permutation is applied in a single epoch according to our implementation.

III. Baseline Implementation

A. Overview

Santa Cruz, Rodrigo, et al. proposes DeepPermNet to solve the jigsaw puzzle problem. [1] This network consists of an extractor and an aggregator, which outputs a $k \times k$ matrix. Then Sinkhorn-Knopp algorithm is applied to generate a doubly stochastic matrix for prediction. The model construction is roughly shown in the following figure.

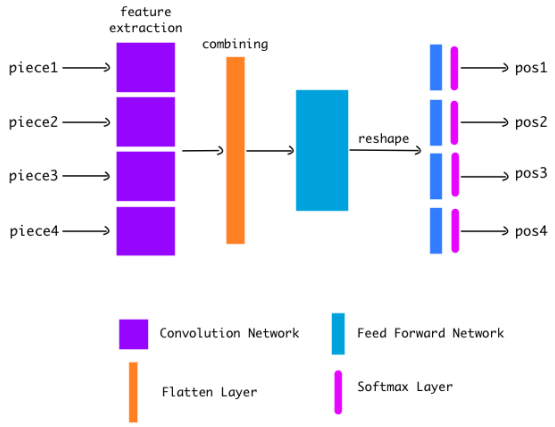

Fig. 2. network architecture

Each sub-image shares the same extractor, which provides a 512-dimensional feature vector. Then the aggregator will combine the features and output a $k \times k$ matrix. The details will be revealed in the following subsections.

B. Extractor

To fit the small size sub-images, we design a naïve CNN as the feature extractor for the 2×2 jigsaw puzzle problem, which is shown in the following figure. For the 3×3 case, the network requires a little modification, which is also shown in the figure below. [2]

Whichever the case is, the feature map will be flattened and fed into FC layers to generate a 512-dimensional vector, which is aligned with the input of the aggregator.

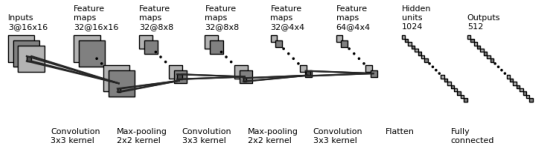

Fig. 3. baseline extractor model (2×2)

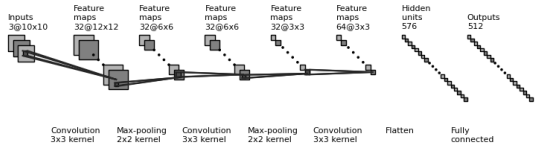

Fig. 4. baseline extractor model (3×3)

C. Aggregator

The feature vectors of the sub-images are concatenated into a single vector and then fed into the aggregator. The aggregator is a naïve FCN, which is shown in the following figure. [3]

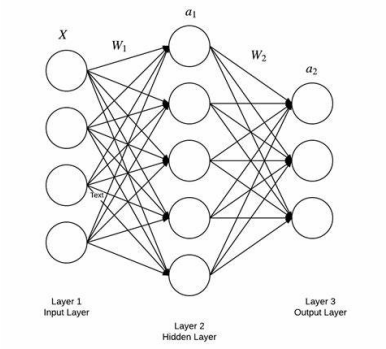

Fig. 5. baseline aggregator model

The input of the aggregator is a $512n$-dimensional vector, and the output is an $n \times n$ matrix. Only one hidden layer is included. The hidden layer is as wide as 4096-dimensional to rapidly aggregate all the features and form an overall estimation. Hence, the aggregator also plays the role of classifier.

D. Sinkhorn Algorithm

Admittedly, the aggregator outputs a matrix with the required shape. However, there is no guarantee for its combination properties. We can convert this problem into a quadratic assignment problem [4], which is specified as

$$\begin{aligned} &\min_{\boldsymbol{X} \in \{0,1\}^{n \times n}} \boldsymbol{X}^T \boldsymbol{M} \boldsymbol{X} \\ &\text{s.t. } \boldsymbol{X} \cdot \mathbf{1} = \mathbf{1}, \boldsymbol{X}^T \cdot \mathbf{1} \leq \mathbf{1} \end{aligned} \tag{3}$$

where $\boldsymbol{M}$ is the affinity matrix generated by the aggregator and $\boldsymbol{X}$ is the permutation matrix we expect. It has been proved that this optimization problem is NP-hard. As an alternative, we try to figure out the doubly stochastic matrix $\boldsymbol{P} \in [0,1]^{n \times n}$. Similarly, it holds that

$$\boldsymbol{P} \cdot \mathbf{1}^n = \mathbf{1}^n \tag{4}$$

$$\boldsymbol{P}^T \cdot \mathbf{1}^n = \mathbf{1}^n \tag{5}$$

which makes $\boldsymbol{P}$ meaningful in terms of probability. Therefore, such $\boldsymbol{P}$ can be taken as the prediction given by the model.

The doubly stochastic matrix can be efficiently computed by Sinkhorn-Knopp algorithm. [5] Sinkhorn algorithm is an iterative procedure. In each iteration, the matrix is normalized by rows and columns respectively, until the matrix converges into a doubly stochastic matrix within an ε-tolerance.

Algorithm 1 Sinkhorn-Knopp Algorithm

Input: Matrix $\boldsymbol{M} \in \mathbb{R}^{n \times n}$, parameter $\lambda \in \mathbb{R}$
Output: Doubly stochastic matrix $\boldsymbol{P} \in \mathbb{R}^{n \times n}$

```
Initialize P = e^{-λM}
while none-convergence do
    Normalize P by rows
    Normalize P by columns
end while
Return doubly stochastic matrix P
```

The convergence and efficiency of Sinkhorn algorithm has been strictly proved. Note that applying Sinkhorn algorithm brings two prominent advantages. First, the output is a doubly stochastic matrix, which is always reasonable as a prediction. Second, the computation is differentiable, which makes the end-to-end training possible.

Just consider the normalization operation. Let $\boldsymbol{M} \in \mathbb{R}^{n \times n}$ denote the affinity matrix. Row normalization is defined as

$$\boldsymbol{M}_{ij}^{(t+1)} = \frac{\boldsymbol{M}_{ij}^{(t)}}{\sum_{k=1}^{n} \boldsymbol{M}_{ik}^{(t)}} \tag{6}$$

Similarly, column normalization is defined as

$$\boldsymbol{M}_{ij}^{(t+1)} = \frac{\boldsymbol{M}_{ij}^{(t)}}{\sum_{k=1}^{n} \boldsymbol{M}_{kj}^{(t)}} \tag{7}$$

Let $\mathcal{L}$ denote the loss, then the partial derivative for row normalization is specified as

$$\frac{\partial \mathcal{L}}{\partial \boldsymbol{M}_{pq}^{(t)}} = \sum_{i=1}^{n} \frac{\partial \mathcal{L}}{\partial \boldsymbol{M}_{pi}^{(t+1)}} \left[\frac{\mathbb{1}_{i=q}}{\sum_{j=1}^{n} \boldsymbol{M}_{pj}^{(t)}} - \frac{\boldsymbol{M}_{pi}^{(t)}}{\left(\sum_{j=1}^{n} \boldsymbol{M}_{pj}^{(t)} \right)^2} \right] \tag{8}$$

The partial derivative for column normalization can be specified in a dual form, which is omitted here.

E. Inference

On the basis of previous discussion, the target is to find the doubly stochastic matrix $\hat{\boldsymbol{P}}$ such that

$$\hat{\boldsymbol{P}} \in \arg\min_{\boldsymbol{P}} \|\boldsymbol{P} - \boldsymbol{Q}\|_F \tag{9}$$

where $\boldsymbol{Q}$ is the ground truth given by the label and $\|\cdot\|_F$ is the Frobenius norm, which is defined as

$$\|\boldsymbol{A}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{A}_{ij}^2} \tag{10}$$

Hence, we can use the normalized $\|\cdot\|_F^2$ as the loss function, which is specified as

$$\mathcal{L}(\boldsymbol{P}, \boldsymbol{Q}) = \frac{1}{n^2}\sum_{i=1}^{n}\sum_{j=1}^{n} \left(\boldsymbol{P}_{ij} - \boldsymbol{Q}_{ij}\right)^2 \tag{11}$$

which is exactly the same as the MSE loss of $\boldsymbol{P}$ and $\boldsymbol{Q}$. Then the final predictions are given by

$$p_i = \arg\max_j \boldsymbol{P}_{ij} \tag{12}$$

where p_i denote the prediction of the i-th sub-image.

IV. Baseline Result

A. Metrics

Santa Cruz, Rodrigo, et al. applies Kendall's τ, Hamming similarity and l_1 normalization error respectively to evaluate the performance of the model. [1] These metrics attend to evaluate from a perspective of permutation, but fail to describe the spatial structure of the sub-images.

Apart from epoch loss, we apply fragment accuracy and puzzle accuracy for evaluation. The fragment accuracy describes the proportion of the sub-images that are placed in the correct position, which is specified as

$$\tau_F = \frac{1}{mn}\sum_{k=1}^{m}\sum_{i=1}^{n} \mathbb{1}\left[p_i^{(k)} = q_i^{(k)}\right] \tag{13}$$

where $p_i^{(k)}$ is the prediction of the i-th sub-image in the k-th puzzle, $q_i^{(k)}$ is the corresponding label and $\mathbb{1}[\cdot]$ is the indicator function. The puzzle accuracy describes the proportion of the puzzles that are exactly solved, which is specified as

$$\tau_P = \frac{1}{m}\sum_{k=1}^{m} \mathbb{1}\left[\boldsymbol{p}^{(k)} = \boldsymbol{q}^{(k)}\right] \tag{14}$$

where $\boldsymbol{p}^{(k)}$ is the prediction of the k-th puzzle, $\boldsymbol{q}^{(k)}$ is the corresponding label. Note that m is the number of puzzles and n is the number of fragments in a single puzzle.

In general, we expect a high puzzle accuracy, which directly reflects the ability of the model to solve the puzzle. However, there is a rough estimation specified as

$$\tau_P = \frac{1}{m}\sum_{k=1}^{m}\prod_{i=1}^{n} \mathbb{1}\left[p_i^{(k)} = q_i^{(k)}\right] \approx \tau_F^n \tag{15}$$

which indicates that a perfect solution is more than difficult to achieve. Under this circumstance, we turn to expect a high fragment accuracy, which implies the same target as the puzzle accuracy. When the puzzle accuracy is extremely low, the fragment accuracy will be a relatively reliable metric.

B. Performance

Based on the previous discussion, we implement a baseline model and train it on the 2×2 and 3×3 datasets respectively.

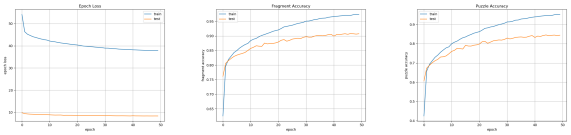

Fig. 6. training curve of baseline on 2×2 dataset

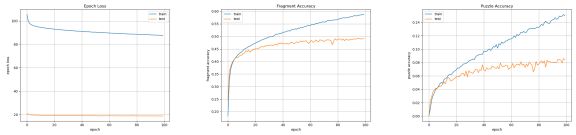

Fig. 7. training curve of baseline on 3×3 dataset

The training curves are shown in the figures above. On the 2×2 dataset, the training converges after about 50 epochs and the model performs a relatively high accuracy. In the 3×3 case, however, the training converges after about 100 epochs.

TABLE II
BASELINE PERFORMANCE

Scale	Set	$\mathcal{L}_E$	τ_F	τ_P
2×2	Train	37.8307	97.30%	95.11%
	Test	8.3484	90.72%	84.43%
3×3	Train	87.6979	58.75%	14.99%
	Test	18.7751	49.20%	8.44%

We can see that on the 2×2 dataset, the fragment accuracy is 90.72% and the puzzle accuracy reaches 84.43%, which is satisfactory to some extent. On the 3×3 dataset, the fragment accuracy is only 49.20% and the puzzle accuracy falls to 8.44%, which is a failure. Such result is not surprising since the 3×3 dataset is much more difficult than the 2×2 dataset.

We notice that the feature extractor tends to learn the semantic information of the fragments and ignore the visual continuity. Such behavior may be useful for pretext tasks, but will lead to severe performance degradation in the pure jigsaw puzzles. The solution will be discussed in the next section.

In addition, we notice that the training is extremely slow, which is mainly caused by Sinkhorn algorithm. Although Sinkhorn algorithm has a low time complexity, the iterative procedure is still time-consuming and brings much more computation for back propagation. We will try to solve this problem in the next section.

V. Model Improvement

A. Data Augmentation

Although the dataset is large enough, the quality of the sub-images is a problem. Consider the 3×3 dataset, the sub-images are equally cropped and has only 10×10 pixels. It is impossible for the extractor to learn all the key features from such small sub-images. Therefore, we introduce a transformation, where all the sub-images are resized to 32×32 pixels and then converted to tensors. This transformation brings two benefits. First, more details are preserved in the sub-images, so the extractor can learn both the semantic information and the visual continuity better. Second, the inputs are aligned, so we no longer need to design different extractor models for different scales.

B. Network Architecture

Since the shape of the inputs has changed, we need to make a little modification to the feature extractor.

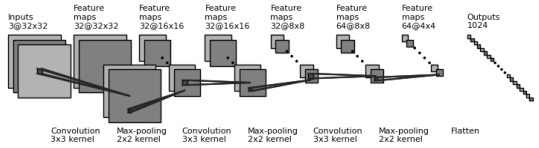

Fig. 8. improved extractor model (universal)

We can see that another max pooling layer is added and the features extracted becomes 1024-dimensional, which promises a better feature representation for the puzzle solver. Accordingly, the input of the aggregator becomes $1024n$-dimensional. An 1024-dimensional hidden layer is added to enhance the representation ability of the aggregator. Note that the output is still an $n \times n$ matrix.

Considering the training efficiency, we only enable Sinkhorn algorithm on the test set to maintain the rationality of the predictions. Admittedly, this will lead to some loss of interpretability, but the training speed is significantly accelerated.

C. Loss Function

If we still use the MSE loss function, a sigmoid or softmax activation function is required to normalize the output, which violates the intention of removing Sinkhorn algorithm from the training process.

We might as well consider the jigsaw puzzle problem as a classification task [6], where the i-th row of $\boldsymbol{P}$ provides a probability distribution of the i-th fragment. Then we can use the cross entropy loss function, which is specified as

$$\mathcal{L}(\boldsymbol{P}, \boldsymbol{Q}) = -\sum_{i=1}^{n}\sum_{j=1}^{n} \boldsymbol{Q}_{ij} \log \boldsymbol{P}_{ij}$$

where $\boldsymbol{Q}$ is the ground truth given by the label. Lots of previous works have proved that the cross entropy loss function is effective for classification tasks and performs well even without any normalization.

D. Parameter Tuning

Larger sub-images require more memory, so we reduce the batch size to $\frac{1}{4}$. To ensure convergence, we train more epochs and apply a learning rate decay. All the hyperparameters are carefully tuned to achieve the best performance.

VI. Improved Result

With the previous methods, we train the improved model on the 2×2, 3×3 and 4×4 datasets.

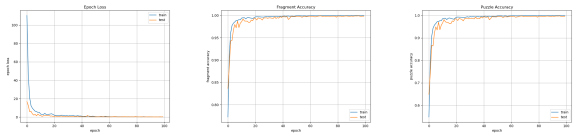

Fig. 9. training curve of improvement on 2×2 dataset

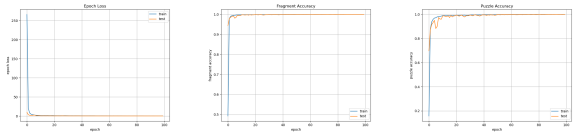

Fig. 10. training curve of improvement on 3×3 dataset

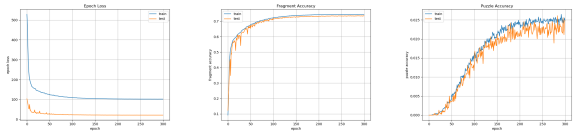

Fig. 11. training curve of improvement on 4×4 dataset

Comparison of model performance between baseline and improvement is shown in the following table.

TABLE III
IMPROVED PERFORMANCE

Scale	Model	$\mathcal{L}_E$	τ_F	τ_P
2×2	Baseline	8.3484	90.72%	84.43%
	Improved	0.2013	**99.84%**	**99.63%**
3×3	Baseline	18.7751	49.20%	8.44%
	Improved	0.0256	**99.98%**	**99.86%**
4×4	Baseline	-	-	-
	Improved	21.4948	**73.55%**	**2.49%**

On the 2×2 dataset, the improved model perfectly solves the jigsaw puzzle problem with a probability of 99.63%, which is extremely successful. On the 3×3 dataset, the improved model achieves a even higher accuracy of 99.86%, which completely outperforms the baseline. Moreover, the improved model correctly places 73.55% of the fragments on the 4×4 dataset, where the baseline does not work at all.

Conparison of computation efficiency is shown in the following figure. We can see that removing Sinkhorn algorithm significantly accelerates the computation. The improved model is approximately 7 times faster than the baseline in both learning and inference stages.

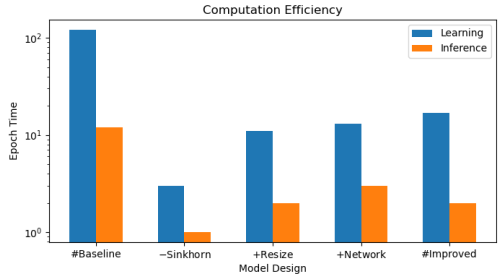

Fig. 12. comparison of computation efficiency

APPENDIX
TRAINING PARAMETER

For the baseline model, the batch size is 1024. We use Adam optimizer with constant learning rate of 10^{-3}. The model is trained for 50 epochs on the 2×2 dataset and 100 epochs on the 3×3 dataset. For the improved model, the batch size is 256. We use Adam optimizer with initial learning rate of 10^{-3} and decay of 0.8 every 10 epochs. The model is trained for 100 epochs on the 2×2 and 3×3 datasets and 300 epochs on the 4×4 dataset. All the experiments are run on a single NVIDIA A100 80G Tensor Core GPU.

APPENDIX
RESULT VISUALIZATION

Here are some jigsaw puzzle solving examples of the baseline model on the 2×2 and 3×3 datasets. The problems are randomly sampled from the test set as a batch of 16 images. The solutions are generated by the baseline model.

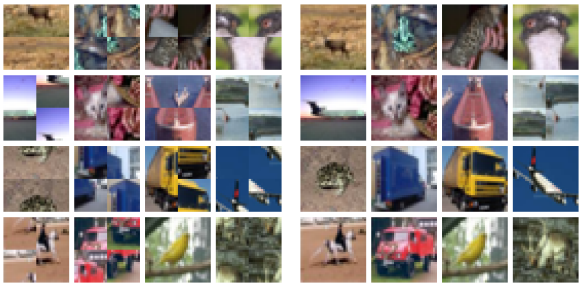

Fig. 13. solving sample of baseline on 2×2 dataset

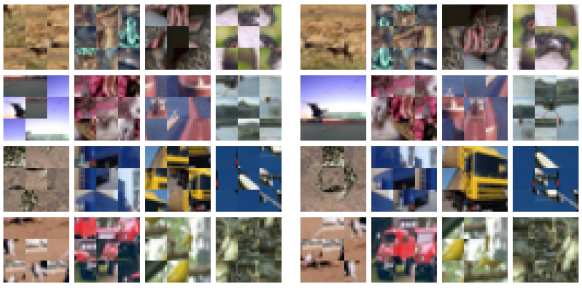

Fig. 14. solving sample of baseline on 3×3 dataset

We can see that the baseline model perfectly solves most of the 2×2 jigsaw puzzles. However, it can only solve a few of the 3×3 jigsaw puzzles. For comparison, here are some jigsaw puzzle solving examples of the improved model on the 3×3 and 4×4 datasets respectively.

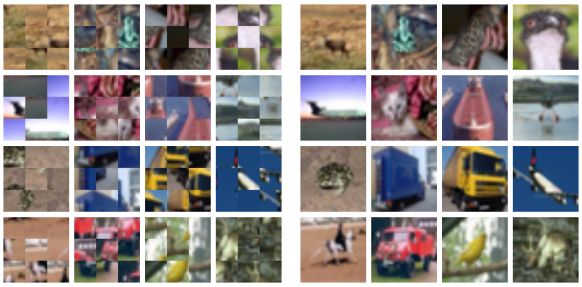

Fig. 15. solving sample of improvement on 3×3 dataset

Fig. 16. solving sample of improvement on 4×4 dataset

We can see that the baseline model perfectly solves all the 3×3 jigsaw puzzles. Only a part of the 4×4 jigsaw puzzles can be correctly solved, but the mistakes seem to be more reasonable to human eyes.

REFERENCES

[1] Santa Cruz R, Fernando B, Cherian A, et al. Deeppermnet: Visual permutation learning[C]. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. 2017: 3949-3957.

[2] LeCun Y, Bottou L, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.

[3] Rosenblatt F. The perceptron: a probabilistic model for information storage and organization in the brain[J]. Psychological review, 1958, 65(6): 386.

[4] Koopmans T C, Beckmann M. Assignment problems and the location of economic activities[J]. Econometrica: journal of the Econometric Society, 1957: 53-76.

[5] Sinkhorn R, Knopp P. Concerning nonnegative matrices and doubly stochastic matrices[J]. Pacific Journal of Mathematics, 1967, 21(2): 343-348.

[6] Paumard M M. Solving Jigsaw Puzzles with Deep Learning for Heritage[D]. CY Cergy Paris Université, 2020.

[7] Chen Y, Shen X, Liu Y, et al. Jigsaw-ViT: Learning jigsaw puzzles in vision transformer[J]. Pattern Recognition Letters, 2023, 166: 53-60.

[8] Doersch C, Gupta A, Efros A A. Unsupervised visual representation learning by context prediction[C]. Proceedings of the IEEE international conference on computer vision. 2015: 1422-1430.

[9] Noroozi M, Favaro P. Unsupervised learning of visual representations by solving jigsaw puzzles[C]. Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part VI. Cham: Springer International Publishing, 2016: 69-84.