Shredder Recovery Challenge

Eyal Ben David - 305057416 Etai Wagner - 302214606

August 3, 2018

Abstract

Shredder reconstruction has many real-life applications from military use to archeology. In this project we consider the square shredder problem, where the goal is to reconstruct the image from a set of non-overlapping, unordered, square shredded parts. We propose a fully-automatic, general solver, which assumes no prior knowledge about the original image and uses deep neural networks. It is general in the sense that it can handle an unknown number of shredded pieces and it can handle images and texts, without prior knowledge.

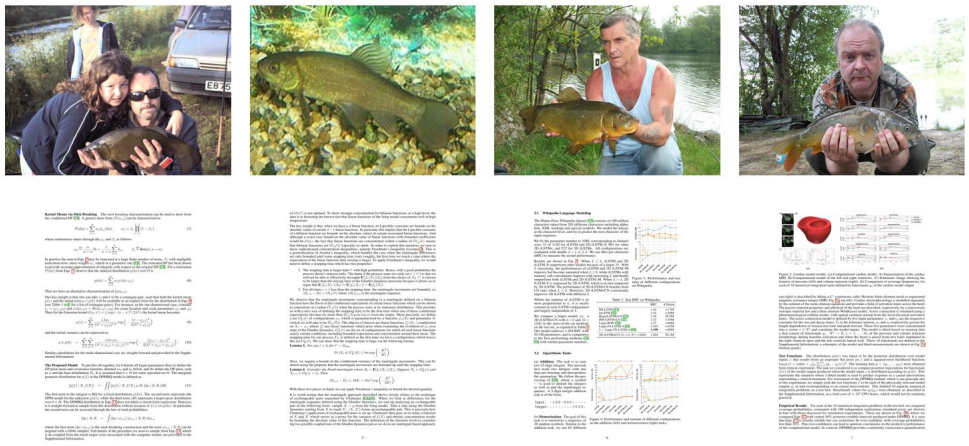


Figure 1: Original dataset examples, texts and images, before they are shredded and transform to grayscale

1 Introduction & Motivation

The "Shredder Recovery challenge" or "the jigsaw challenge" as it is more commonly referred to in literature is a known challenge with many proposed solutions [1] [2] [3]. Yet almost none of the solutions use deep neural networks in their solutions. Attempts have been made to solve the problem using end to end NN with disappointing results [4] [5]. We hypophysis that this is due to the challenge of defining a mathematical loss that will capture the nature of the problem. Never the less we started with a basic NN exploring the data set and receiving initial results. We used a two LSTM layers over a time distributed CNN. Our motivation was that the convolutional layers will extract the relevant features and the LSTM will be able to position them accordingly. This is somewhat similar to translation where word ordering can be switched by the network in order to comprise a more meaningful sentence.

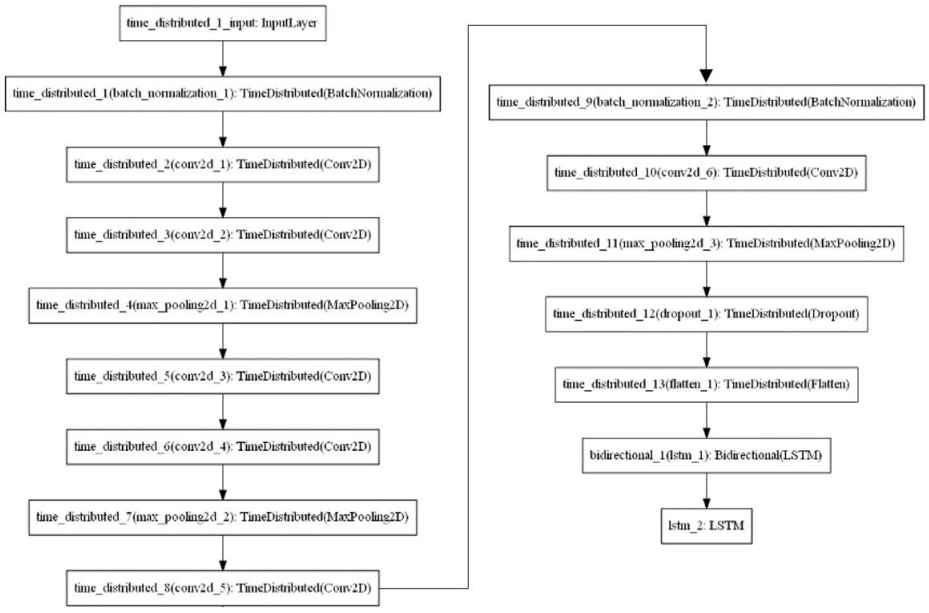


Figure 2: Basic hybrid CNN + LSTM

This approach showed promise achieving very high accuracy around 96% on the training data (for 4x4 puzzles) but failing to generalize well. Validation set accuracy peaking at 46%. We tried different methods to try and improve the result such as:

- different regularizers (dropout, L1, L2)
- attention decoder
- various width and depth

- different activation functions
- constructing our own loss function that takes into account that no position can be selected twice.
- An added cross correlation matrix as prior knowledge
- Feature extraction by a pre-trained network (ResNet, VGG)
- Choosing 1 peace at a time and inputting that decision back into the net

But these methods did not manage to improve the results. We suspect this is due to the very small data set presented which led the network to overfit quickly.

Another major disadvantage of this approach is it dependency on the input sequence length. Different puzzles would need different networks to classify them each network dealing with a different series length. This is a draw back since it harms the generality of the finale solution. Due to all of the above we chose a completely different approach to be described in the next section.

2 Neural Network architectureand Training

We decided to use a more scalable approach meaning a fixed number of NN no matter the sequence length. For our first try we created a model which for two given patches predicts their relations: right, left, up, down or not neighbors. This approach didn't succeed as our NN did not manage to predict correctly on the validation set. Next, we decided to simplify the problem as much as possible by training four different CNN's, Each CNN will receive a pair of image patches and decide if they are neighbors in a specific orientation: left, right, up or down. This is a rather simple task and we expect good results even with the limited training set at our hand. The architecture we chose is a simple CNN where we concatenate the two image patches in the orientation we are testing for. This is then entered into the CNN and the result is a binary classification 1 for neighbors and 0 for not.

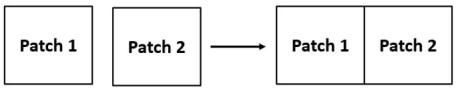


Figure 3: concatenation example for right orientation

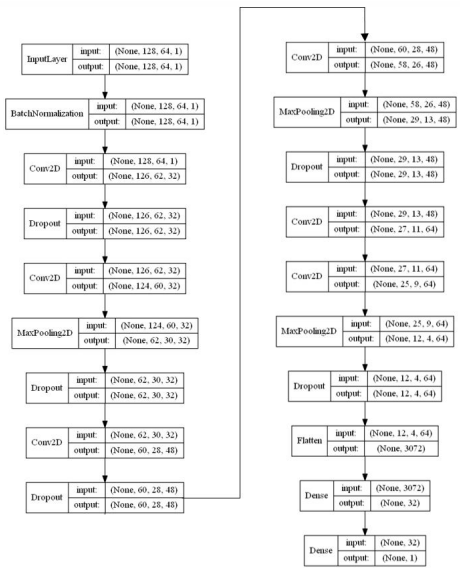


Figure 4: our CNN architecture for down orientation

We trained the CNN on all forms of cropped images together. This greatly increased our training set but presented a major problem. About 94% of the

examples are not neighbors, this inserts a great bias into the training and indeed our first model classified almost all inputs as not neighbors, yielding a lot of false negative classification. To overcome this obstacle, we created a custom loss:

$$(1)\ -3y_{true}log(y_{pred}) - (1 - y_{true})log(1 - y_{pred})$$

This is the regular binary cross-entropy with a bigger loss for miss detected neighbors. This approach managed to counter the bias entered by the uneven data set achieving high accuracies with minimal miss detects.

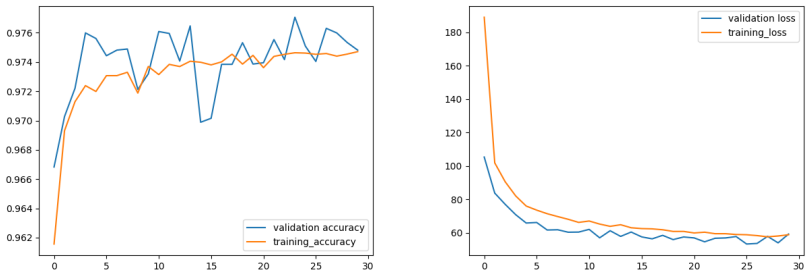


Figure 5: Training graphs

3 Differences Between Texts And Images

As in real life, the best ways to reconsruct a text puzzle and an image puzzle are different. Text have noticable frames and preferable rows connectivity over columns conectivity. Furtheremore in frequency domain text use all frequencies, as in space domain it contains lots of steps. It is impossible to satisfy the nyquist criterion for text images, Hence we sample them with a higher rate, to reduce the information we lose. On the other hand, images use only low frequencies and are smooth. Because the smootheness of images does not rely on the orientation, rows conectivity is not preferable over colomns conectivity. As long as we satisfy the nyquist criterion, we do not lose any information, so it is useless to sample in a higher frequency.

4 Algorithm Outline

Once we trained all our networks, the assembly algorithm is crucial. If you misplace a piece in the early stages of ordering you won't get a good result. Hence sometimes choosing the first piece is crucial. Our algorithm is comprised of three steps:

1. Determine if the picture is a text or an image.

2. Calculating four compatibility matrices between all pieces, one for each orientation.

3. Choosing the first piece

4. Assembly

4.1 The compatibility metric

A good compatibility metric is a fundamental component for our puzzle solver algorithm. Given two pieces p_i, p_j and an orientation left, right, up, down, the compatibility function predicts the likelihood that p_i and p_j are neighbors in the spatial relation. We first use our trained CNN to evaluate the relations between all pieces for every orientation to get four matrices P:

(2) $P(i,j,r) = evaluate(concat(i,j,r))$

We then use matrices P to calculate dissimilarity matrices D, where $D(i,j,r)$ represent the dissimilarity of piece 'i' to piece 'j' at a given prientation 'r'.

(3) $D(i,j,r) = 1 - P(i,j,r)$

Finally we compute compatibility matrices for each orientation C, where $C(i,j,r)$ represents the compatibility between piece 'i' and piece 'j' at a given orientation 'r'.

(4) $C(i,j,r) = 1 - \frac{D(i,j,r)}{second(D(i,r))}$

Where $second(D(i,r))$ is the second most highest value of row i in the matrice D_r.

4.2 Best Buddies

Because the algorithm is greedy, deciding that two pieces are neighbours is very risky, because once the decision has been taken, there is no way back. Hence we would like to increase the probability of not making early mistakes. In order to do so we define best buddies. Two pieces are best buddies only if they both consider the other piece to be their closest neighbor in a given direction. From our own experiment, if two pieces are best buddies in a given direction, we can assume they are real neighbours in the given orientation with high probability.

4.3 Nearest Neighbours

Because most times we cant place all pieces using only best buddies method, we use also a nearest neighbur method. Piece 'j' is the nearest neighbur of piece 'i' if it has the highest value in row 'i' in the given matrix C_r.

4.4 Text or Image

As disscused above, two different algorithms handle texts and images. We have build a 3 layers basic CNN which give very high accuracy predictions. At start of the algorithm we first use this network in order to decide which algorithm to use.

4.5 Choosing The First Piece

As hinted earlier, selecting a first piece is crucial because the algorithm is greedy, hence early mistakes may lead to later failure. We are using a best buddy method to choose the first piece. We are looking for a piece that has the maximum best buddies to be the first piece to start with.

4.6 Assembley

The final step of our algorithm is to assembly all pieces together. It is done using a greedy algorithm while holding a best buddy pool for all chosen pieces. It iterates while maintaining a pool of best buddies and a pool of nearest neighbors in case there are no more non chosen best buddies.

4.7 Algorithm

Our Algorithm is greedy and relies on our compatibility matrices. It maintains a pool of candidate pieces to be placed, starting with best buddies pool and then continues with nearest neighburs pool. It iterates on placing a piece from these pools and then adding candidates to it. We elaborate below.

Algorithm 1 Placer Algorithm

```
1: while best buddies pool is not empty do
2:     Extract highest best buddies probability
3:     Place the chosen piece in the puzzle
4:     Find the best buddies of the new piece and add them to the best buddy's
   pool
5: end while
6: if there are more pieces in non-chosen pool then
7:     Extract nearest neighbors pool of all chosen pieces
8:     while not all pieces were placed do
9:         Extract the highest nearest neighbor probability out of the pool
10:        Place the chosen piece in the puzzle
11:        Find the nearest neighbors of the new piece and add them to the
   pool
12:    end while
13: end if
```

Experiments had yield the conclusion that text rows probabilities are much more reliable than those of the colomns. Hence in our algorithm, while reconstructing text images we give higher weight factor for rows probabilities. This is our way of building a full row before descending or ascending a row.

5 Results and Conclusions

We performed our testing on six images three documents and three images shredded into different length to simulate the upcoming test our results:

We get 100% on 2x2 puzzles.

(a) text input

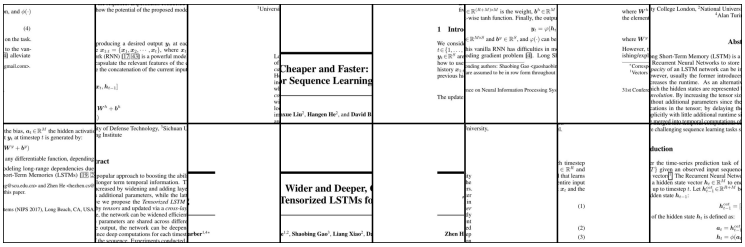
(b) image input

Figure 6: 2x2 shredded inputs

(a) Original Images (b) Reconstructed Images

Figure 7: 2x2 test images

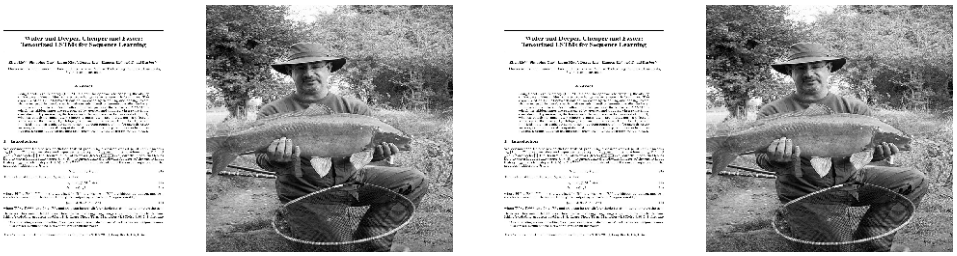
We get 100% on 4x4 puzzles.



(a) text input

(b) image input

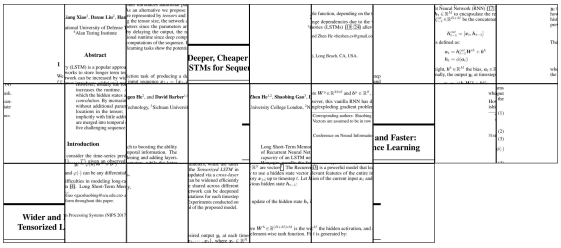
Figure 8: 4x4 shredded inputs



(a) Original Images (b) Reconstructed Images

Figure 9: 4x4 test images

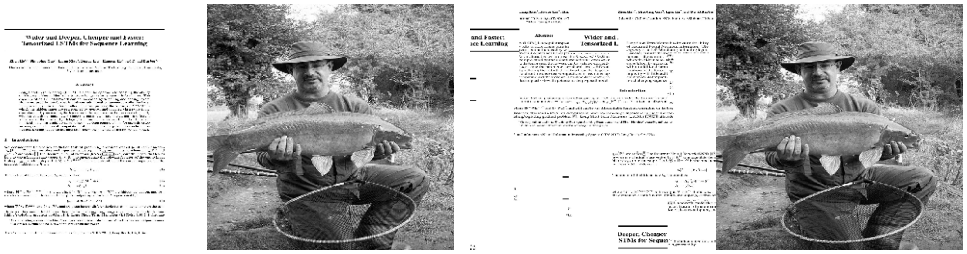
We get 50% on 5x5 puzzles.



(a) text input

(b) image input

Figure 10: 5x5 shredded inputs



(a) Original Images (b) Reconstructed Images

Figure 11: 5x5 test images

The Shredder problem is a complicated one not clearly suited for deep neural networks. This is mainly do to the fact the problem isn't defined mathematically very well hence defining a loss function is tricky. Never the less our solution solves the problem with high accuracy for all the shredding possibilities in minimal time. Our solution is not bounded by the number of shredded pieces and is capable of preforming well for any number of pieces.

6 References

[1] G. P. Ayellet Tal, "Solving Multiple Square Jigsaw Puzzles with Missing Pieces".
[2] O. D. a. N. N. D. Sholomon, " A genetic algorithem-based solver for very large jigsaw puzzles.," 2013.
[3] M. S. a. O. B.-S. D. Pomeranz, "A fully automated greedy square jigsaw puzzle solver," 2011.
[4] R. M. O. A. Lucio Dery, "Neural Combinatorial Optimization for Solving Jigsaw Puzzles".
[5] A. G. N. P. P. R. Viveka Kulharia, "Neural Perspective to Jigsaw Puzzle Solving".