# Deep Learning - 236606 – Final Project

## Shredder Recovery Challenge

Ido Glanz - 302568936 Matan Weksler – 302955372

a. **General -** The projects goal was to design and implement a DNN based algorithm capable of reconstructing pictures and documents cropped to t^2 equal crops. The algorithm had to also to withstand Out of Distribution crops (i.e. belonging to a different picture) and rule them out outputting the correct crop order.

b. **Main Challenges (as we saw them) -**
   i. **Unknown number of crops -** figure could be cropped to 4,16,25 or even more crops and the net had to handle it without any prior knowledge.
   ii. **Varying figure sizes -** even before the different crop sizes, each figure had a different resolution to start with which complicated the input dimensions.
   iii. **Processing both pictures and documents**
   iv. **Output parsing -** assuring all "crop positions" were allocated (e.g. each input crop was **assigned to a different position** in the assembled picture)

c. **Training data generation**
   i. The first challenge was to generate a diverse and broad enough dataset to train on, incorporating OOD crops, labeling each set and finally shuffling them together.
   ii. At first, we had to bound our dimensions and align all crops internally (number of pixels per crop) as well as the max number of crops. We decided to start with resampling each crop to 40x40 shape (down or up sampling as needed, we tested multiple resample sizes and at a certain point we ran into too many OOM issues and confined to 40by40 which showed the best results given the computing power we had) and generating up to a max of 5x5 crops + 5 OOD crops, hence a max of 30 crops. Figures cropped to less shreds were padded to 30 crops (with zero matrices). We assume that with more data and more training time this could be generalized to larger sizes and we wanted to bound the problem for a reasonable training time and complexity.

iii. Our shredder then randomly voted the number of cuts (t), ranging from 2 to 5 per picture or document and generated the labeled data set (crop set and label matrix which will be discussed next). This was done on the entire picture and document set numerous time to generate from each picture sets of different crop sizes and of different shuffle order. For the final runs, we generated approx. 30K different sets.

iv. **Data labeling -** this was a big challenge as we wanted each crop to be voted to a certain location in the assembled figure also taking into account whether a different crop suited better to that location (hence soft-maxing not just over each crop but over each location).

To start with, we generated an M by N matrix, where M is the max number of crops (e.g. 30 for a 5x5 shredded figure along with 5 max OoDs) and N is the max number of belonging crops + a column for OoDs and a column for padding (hence a total of 27 columns).

Reading a bit, we found a few articles on the usage of Sinkhorn matrix normalization as a mean to normalize a matrix as so each row and column have one max value (Doubly Stochastic Matrices). Basically, this is done be running a softmax norm on the rows and then the columns for a few rounds which (not always) normalizes the matrix in the manner we discussed above.

**d. Net architecture**

i. Essentially, we figured the network should include two stages – the first running in parallel on each crop and extracting an embedding vector (or a features vector) which is independent from the other crops and the general layout of the figure but generates (we'd assume) features relevant for predicting adjacency of crops and relative positioning. For this task, we implemented a CNN with multiple layers (see figure 1) running in parallel (Time-Distributed) on all crops and generating a 256 element features vector.

Second, to predict the crops' position in the complete figure, we figured an RNN would be best suited to grasp the sequential nature of the picture and the relations between figures. To try and be more robust to ordering we used bidirectional LSTM layers (3 in total) as so the last layer $n^{th}$ LSTM cell would output a probability vector over the position of the $n^{th}$ crop.

The output matrix from the net is hence a $[t^2 + t]\ by\ [t^2 + 2]$ matrix: max possible crops (including max OoDs) by relative locations + label for OoDs and padding.

ii. **General algorithm pipeline:**

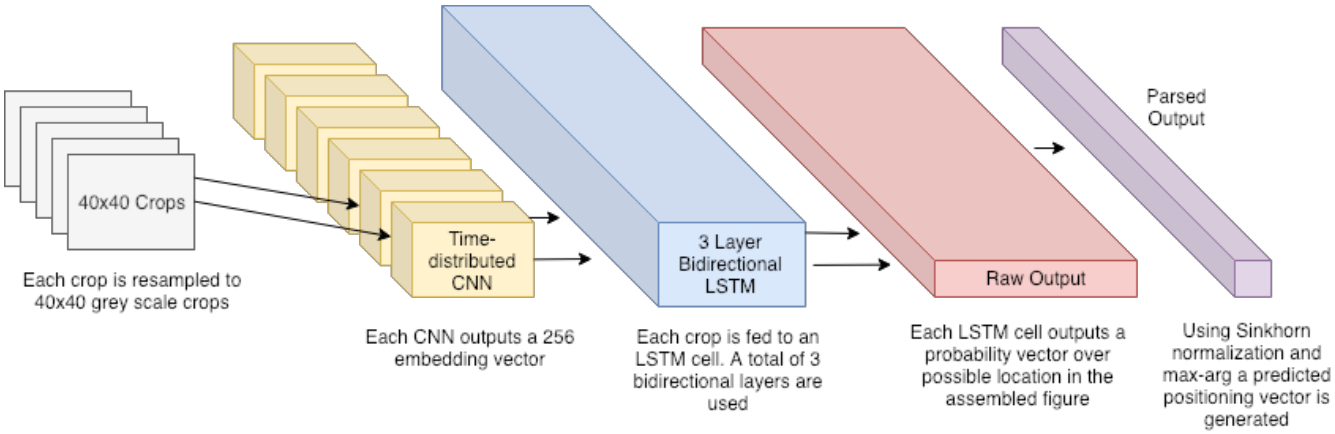


*Figure 1 - algorithm general pipeline.*

iii. **CNN side:**

For the CNN part of the network we wanted to allow enough layers to generate features able to later assist in predicting relative locations and converge to a reasonable size vector for the LSTM to input. We partially based the network on the VGG16 architecture (only starting with smaller input size and less layers). We ended up using 6 convolutional layer and max pooling, and while we started with a lot of drop out, we ended up removing them due to a suspect they were causing the network to converge to a local minimum as so all crops are OODs or padding. To our understanding, the added massive regularization limited the model to be complex enough for the task and encouraged it to predict all crops as either OODs or padding. This proved right running for t= 2, 3 generating higher accuracy but we can't say for sure it did the job for t= 4-5.

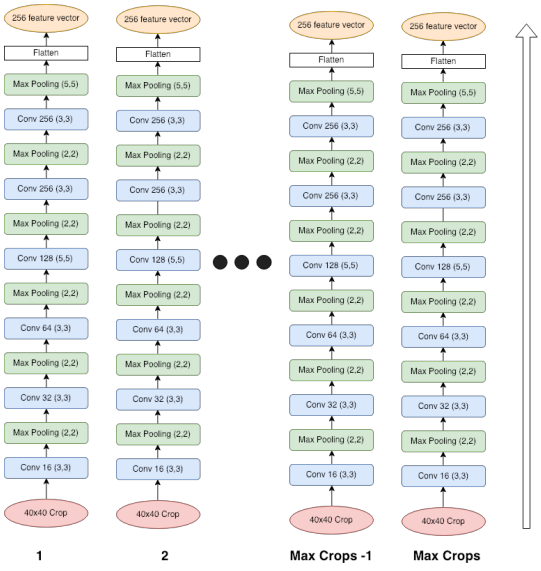


*Figure 2 - CNN architecture. Partially based on VGG-16*

iv. **RNN side:**

For the RNN side of the network, our first consideration was to have it bidirectional as so it can grasp relative location from both sides and not be too depended on the shuffling (also based on related articles, see 3). We also added 3 consecutive layers of such cells as so the net is deeper and able to generate more permutations (we think) until converging to the maximal one.

As we used multiple layers we returned the sequences in all layers while on the last one we used it as the output of the overall network.

See figure 3 for the RNN side architecture.

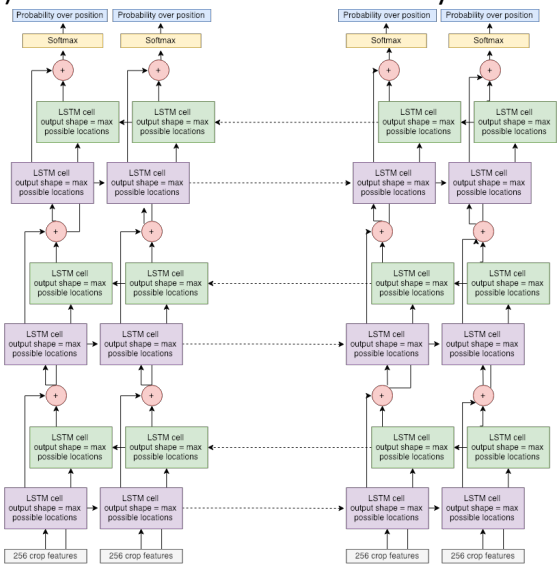


*Figure 3 - RNN architecture. We used bidirectional LSTM trying to overcome input ordering*

**e. Output parsing:**

i. As mentioned above, the output from the network is a max_possible_crops by n_possible_locations+2 (OoDs and padding). Hence, we needed a method for intelligently 'max-arging' the array to generate an output vector of locations for the given crops. To start off, we throw out all padded rows (as we know how many crops were found in the input directory) and softmaxed all rows to find the highest crops likely to be OoD (column 26) and throw these out as well.

ii. Next, as also mentioned before, we used the Sinkhorn normalization method and ran a softmax function on all rows and columns for 4 times (now running on a square matrix of the true number of crops, i.e. 4,16,25). We tested this for a few synthetic generated outputs and it seems that 2-3 time did the job but reading in article (1) it was advised to run for more iterations. We settled for 4 but this could obviously be further researched upon. We then choose the max-arg of each row and mark its' index as it's chosen position. Once a position was occupied it was not available for other crops to occupy (in this case as we ran from index 1 and on, smaller numbers got

priority - this also could have been done randomly but we had bigger issues to tackle and didn't get to it).

**f. Training**

After (many) hours of trying to work with one single net for up to 25 (+5 OoDs) max crops, padding if needed (i.e. a figure shredded to 9 crops would be padded with 21 zeros figures), we kept reaching a max accuracy of 50% which if looking deeper is actually the net getting **stuck in a local minimum predicting all crops to either be OoD or padding** (on average 50% of crops are indeed padding or OoDs).

In aim to try and overcome this, we first tried removing almost all regularization and dropout as so the model isn't "held back" and can get more complex. After this showed some minor improvement, we tried augmenting the labeling of OODs and padding as so they don't contribute to the loss function as much with the logic of not having this big of a local minimum and having the net converge there too easily. This actually helped in this sense but still we weren't able to reach over 35% accuracy.

**g. Alternatives**

i. Trying to overcome the above, we tried to think of other approaches for the task, and the two main ones we thought of (and partially tested) were training a different network for different sizes of crop numbers (i.e. for t=1,2,3 etc.) and having the algorithm assign the relevant network. This proved well for 2x2 shreds as can be seen in figure 4 but still for greater number of shreds (4-5) wasn't too much of an improvement. Though this did show potential, we felt the breakdown was limited and not scalable and we went back to trying a single net to rule them all.

ii. A second approach we tested was using a fully convolutional network able to generate permutations of the input crops searching for an optimal one. This though didn't show promising results and though we tried to have the CNN generate different permutations of the arranged crops, we felt it couldn't generate a model diverse enough to grasp the complexity of the task (especially incorporating the OODs which were hard to handle in this method).

**h. Results:**

i. Finally, we reverted to using a single network to try and predict crops of sizes 2,4,5 - as said before, we assume the generalization to 3 and 6 possible with more time and memory while for greater numbers it should be tested. We ran both the pictures and documents together (a total of 50K sample sets) and our best results were approx. 40% accuracy. Not too good but at this point we ran out of time and money. Further work could be done on both generating a greater data set, running on bigger batches (we had to limit our learning to 32 as we ran into OOM problems on the server) and also optimizing the weight decay and regularization values (within the CNN).

ii. One of the main challenges we encountered was the great number of tunable parameters and the wide range of degrees of freedom, together with the quite long running times to test different models.

iii. For the 2x2 shredded figure, the net proved to be more capable and managed to achieve around 70-78% accuracy (see figure 4) whilst enlarging it to 4x4 was more complex and was harder for the network to grasp and assemble.

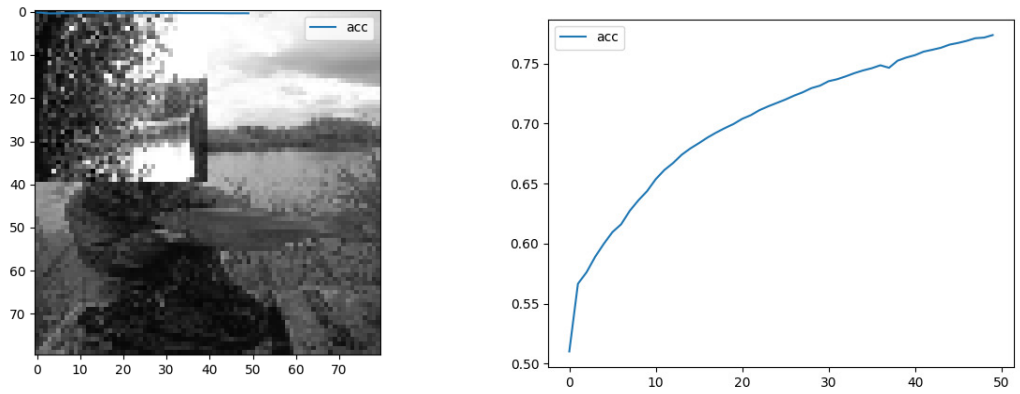


*Figure 4 - 2x2 recovered picture (approx. 75% accuracy on the validation set)*

iv. Finally, as we thought this was a better goal to pursue, we tried to teach the network to learn figures cropped to 2,3,4,5 and have it learn it all together (meaning both picture and documents and with a lot of padding) and see if it could converge to reasonable results. Unfortunately, we ran out of time (and money) and we managed to achieve around 60% accuracy which we are not certain is not a local minima of paddings and OoDs (needs further work).

v. As the task is indeed a complex one, assuming more computing power we would try deeper networks, a lot more data (different shredding and shuffles), larger batch sizes and iterate over different parameters to get a “feel” on their relative effect.

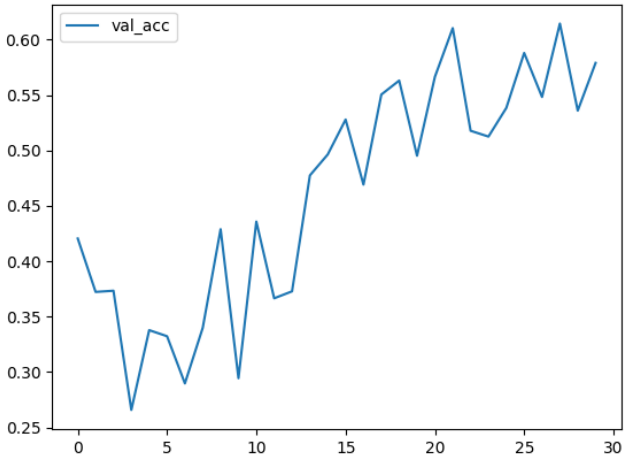


*Figure 5 - Validation accuracy of a single network solution. This needs further work as padding and OoDs seem to generate local minimums.*

**References:**


1. Mena, Gonzalo, et al. "Learning latent permutations with gumbel-sinkhorn networks." *arXiv preprint arXiv:1802.08665*(2018).

2. Sinkhorn, Richard, and Paul Knopp. "Concerning nonnegative matrices and doubly stochastic matrices." *Pacific Journal of Mathematics* 21.2 (1967): 343-348.

3. Dery, Lucio, Robel Mengistu, and Oluwasanya Awe. "Neural Combinatorial Optimization for Solving Jigsaw Puzzles: A Step Towards Unsupervised Pre-Training."

4. Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." *arXiv preprint arXiv:1409.1556* (2014).