Neural Combinatorial Optimization for Solving Jigsaw Puzzles: A Step Towards Unsupervised Pre-Training

Lucio Dery
Stanford University
ldery@stanford.edu

Robel Mengistu
Stanford University
robel@stanford.edu

Oluwasanya Awe
Stanford University
oawe@stanford.edu

Abstract

This paper presents a combinatorially efficient Neural Network based solution for approaching the Jigsaw puzzle problem. We formally define the Jigsaw Puzzle problem as follows: given an image that has been divided up into evenly-sized pieces and shuffled, we seek to reconstruct the original image from these pieces. This task is non-trivial since the solution space of an $N-$piece puzzle is $N!$. State-of-the-art Neural Network approaches to solving this task skirt the combinatorial solution space by only predicting a subset of solutions. By combining a visual feature extraction pipeline with a Pointer Network for combinatorial reasoning, this project proposes a Neural Network to reassemble arbitrary Jigsaw Puzzles of any size configuration. Using our best network architecture and hyper-parameter configuration our network is able to achieve 77% accuracy on 2×2 class of puzzles and 49% on the 2×3 puzzles. Furthermore, by solving the jigsaw puzzle task as a pretext task which requires no manual labeling, we show that parts of our network can be re-purposed to help solve classification problems. Thus, our network serves as both a Jigsaw Puzzle solver and an Unsupervised pre-training mechanism.

1. Introduction

Jigsaw Puzzles are a ubiquitous class of puzzles and have been present for the last few centuries. The variety of tasks involved in solving Jigsaw Puzzles – detecting the piece, clustering similar pieces and finding adjacent pieces – make this problem an interesting one to solve. Additionally, the combinatorial explosion of even a small Jigsaw puzzle presents it as an important vision task for Neural Networks. For example, a 7×7 puzzle has $49! \simeq 6 \times 10^{62}$ possible solutions.

The Jigsaw puzzle problem has potential applications in recovering shredded documents and reconstructing archaeological artifacts amongst others. Though there are other approaches to solving this problem such Genetic Algorithms (GA), we seek discover a neural network-based approach. Since solving this task involves a lot of spatial reasoning, we hope to show that the networks we train could be used in Transfer Learning for other tasks that require spatial reasoning. Before delving into the approach to this problem, we define the task formally.

1.1. Problem Definition

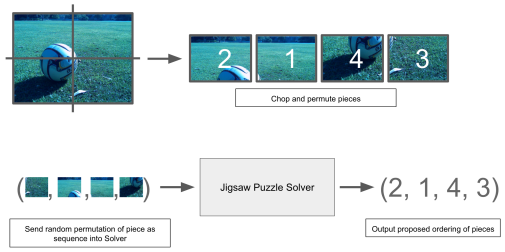

Figure 1: A high-level representation of our problem.

Our Jigsaw Solver will take as input an arbitrary image split into $N = H \times W$ non-overlapping, equally-sized pieces in the same orientation as the original image. Specifically, we can define this input as

$$I = (c_1, c_2, \ldots c_N)$$

where I is the stacked representation of the N pieces. our model will return an ordering, X such that

$$X = (x_1, x_2, \ldots, x_N) = \sigma(1, 2, \ldots, N)$$

such that x_i is the index of the image piece that should be at the ith position in the proposed solution. This ordering, in conjunction with the input, is sufficient to reconstruct the network's solution (see Figure 1).

Our main contribution in this paper is to provide a first step towards tractably solving combinatorial type problems in the space of Computer Vision. In summary, to tackle the Jigsaw Puzzle problem, we propose using a pretrained

model (VGG [10] or Resnet [21]) as a Feature Extractor to generate embeddings of each c_i and pass this embedding though a Pointer Network [13] that outputs an ordering of this input sequence. We discuss this later in detail.

2. Related Work

A variety of work has been done in the field of solving Jigsaw Puzzles using different approaches ranging from Genetic Algorithms [4] to Linear Programming [16]. One of the stronger results was reported by Sholomon *et al*. in [20]. This solution used the Genetic Algorithm paradigm with a novel crossover function that was able to guarantee a better 'child' solution given two 'parent' solutions to the problem. Their current solution produces solutions between $80 - 90\%$ accuracy with piece numbers as large as 3300. Son *et al*. improved on Sholomon *et al*.'s solution using 'loop constraints' by iteratively building loops of puzzle pieces that are consistent [22]. They reported accuracies of around $90 - 96\%$. Another approach to solving this problem was proposed by Yu *et al*. who formulated this problem as a Linear Program-based approach to this challenge. They were able to match that state-of-the-art with accuracy scores [28].

Whilst these solutions give very high accuracies, they are not feasible for real time deployment since they are slow at test time. Given their recent successes at tackling problems in a variety of fields [15] [2] [17] [19] we are convinced that Neural Network based approaches, if fully developed, have the potential to achieve similar accuracies and drastically speed up test time performance since evaluation on a puzzle would involve a single pass.

One of the key ideas in training Neural Networks is the concept of Unsupervised-Pretraining [7]. In unsupervised pre-training, Neural Networks are primed by first solving a task on the data that does not require labels. Noroozi and Favoro [18] attempted a solution to the Jigsaw Puzzle Problem as a means of learning unsupervised representations as pre-training for image classification tasks. They tackled 3×3 puzzles and address the issue of the large solution space by sampling a subset of 100 permutations from the solution space of 362880 selected based on their maximizing Hamming Distance between the 100 configuration. Whilst this approach allowed them to prove that the Jigsaw Puzzle problem could be used as a pre-training task, it does not actually solve the Jigsaw Puzzle Problem.

Recent work by Vinyals *et al*. [26] introduced a new class of Recurrent Neural Network Architectures called Pointer Networks. These Networks are able to tackle combinatorial type problems. By exploiting the attention mechanism [27], Pointer Networks make predictions over the inputs at every time-step instead of attempting to explicitly predict a member of the factorial solution space. In their paper, they were able to achieve then state-of-the-art performance on the Traveling Salesman [12] and Convex Hull problems [6]. Our paper seeks a novel application of their architecture to the Jigsaw Puzzle Problem.

3. Methods and Architecture

Given an $H \times W$ puzzle, our model takes as input a sequence of $N = H \times W$ pieces:

$$X = (x_1, x_2, \ldots, x_N)$$

Where each x_i is an individual image puzzle piece. We seek to learn a function $\pi(X)$ that takes X as input and outputs a permutation

$$\pi(X) = \sigma(1, 2, \ldots, N)$$

that corresponds to the order in which the input image pieces should be sorted to reconstruct the original image. In order to achieve this, we take a two-staged approach. We begin by using a Convolutional Neural Network Architecture on each of the image pieces x_i to extract relevant visual cues λ_i.

Given that the solution space of the Jigsaw Puzzle Problem is factorially dependent on the number of inputs, we cannot simply use the naive approach of combining our visual feature extractor with a Softmax output layer with size equal to the solution space.

Building on the work of Vinyals *et al*. [26], we take the more tractable approach of using a Recurrent Neural Network architecture to produce an output sequence with elements that are discrete tokens corresponding to positions in an input sequence. Our Pointer Network, P [26] takes in a sequence of visual features

$$\lambda = (\lambda_1, \lambda_2, \ldots, \lambda_N)$$

and outputs our desired sorting permuation

$$P(\lambda) = \sigma(1, 2, \ldots, N)$$

. Figure 2 summarizes our model architecture.

3.1. CNN Architecture

Our visual pipeline consists of a Convolutional Neural Network pretrained on the ImageNet Classification task and a Fully Connected Layer (which we dubbed FC_Jigsaw) that mediates the visual pipeline and Pointer Network. Since time and resources constrained how much data we could train our Jigsaw puzzle solver on, we decided to incorporate a pretrained network as a means of expanding our model's predictive capacity. We experimented with Resnet_50 [10] and VGG_16 [21] pretrained architectures.

We feed each puzzle piece x_i through the pretrained network and extract features from the last layer before any

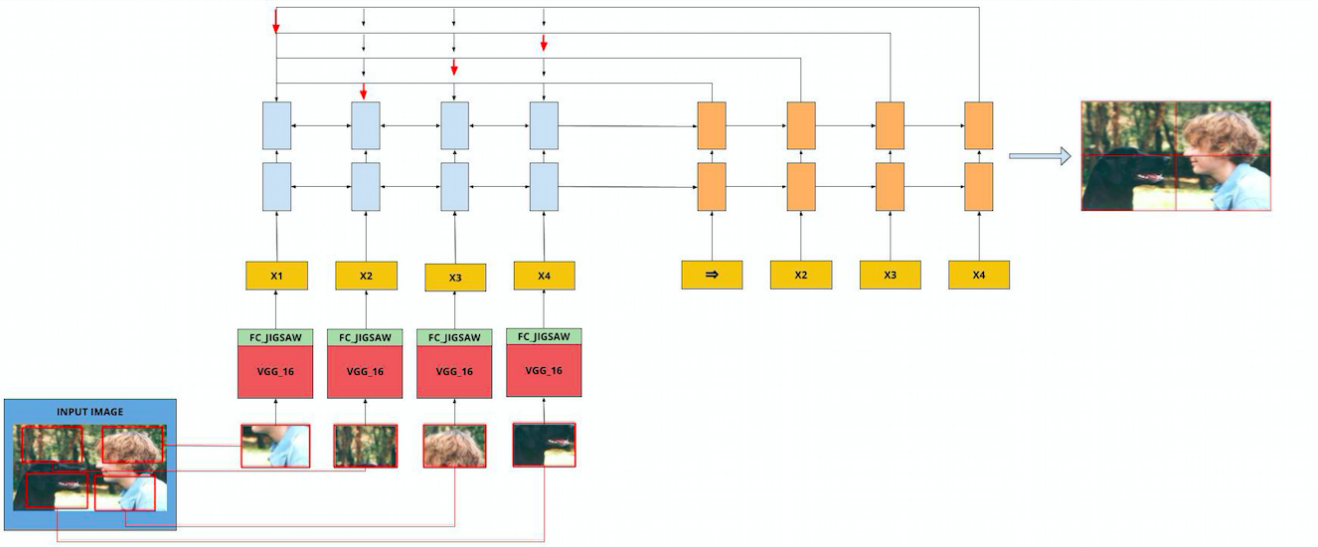

Figure 2: Our architecture for solving jigsaw puzzle problems. Input image chips are projected into a relevant feature space by a CNN visual feature extractor. We then use a Pointer Network to predict a sorting permutation.

fully connected layers. This was necessary since we used smaller image sizes than the networks were trained on. We extracted features from the layer before Resnet_50's softmax layer of size 8192 and the POOL5 layer of VGG_16 of size 2048. Having gotten these embeddings, we pass them through FC_Jigsaw (highlighted in green in Figure 2). The FC_Jigsaw layer was responsible for selecting and transforming the parts of the VGG and Resnet embeddings relevant for solving the Jigsaw problem. It outputs a summary λ_i of the visual features in x_i extracted from VGG and Resnet. We experimented with the output dimension of FC_Jigsaw.

3.2. Pointer Network

Given the set λ of visual summaries of each of the puzzle tiles, our Jigsaw Puzzle solver has to reason across all the tiles to correctly reconstruct the original image. Whilst our visual pipeline thinks locally about each image, joint reasoning is essential to predicting the correct output sequence to reconstruct the image. We solved this part of the problem using a Pointer Network. [26]

A Pointer Network is a Sequence to Sequence model [23] where the decoder chooses or points to a member of the input sequence at each time-step via an attention mechanism [27]. Our Encoder, summarized in Figure 2 is a 2-layer bidirectional Recurrent Neural Network with Gated Recurrent Unit (GRU) cells [3]. We used a bidirectional architecture to make our model more robust to input puzzle ordering. The encoding, $e_j \in \mathbf{E}$, for each visual input λ_j is a concatenation of the forward and backward vectors obtained from the second bidirectional layer. The final hidden state of the encoder is fed as seed to the decoder which decoder maintains a hidden state d_i at each time-step.

Instead of directly predicting a distribution over the combinatorial solution space, the decoder arrives at a solution configuration by giving a probability distribution over the input sequence at each time-step. Let C_i be the distribution over which image piece occupies position i in the correctly reconstructed puzzle.

$$u_i^j = \nu^T tanh(W_1 e_j + W_2 d_i)$$
$$P(C_i|C_1, \ldots, C_{i-1}, \mathbf{E}) = softmax(u_i)$$

W_1 and W_2 are trainable weights of the attention module.

Thus, if e_k is pointed to at time-step i, (i.e argmax$C_i = k$) then it means that the network predicts that puzzle piece k should go into position i in order to reconstruct the image. At time-step $i+1$, we feed the visual embedding of the predicted piece λ_k back into the decoder as input to continue inference.

Our loss function is a time-distributed Softmax loss over the output set $\mathbf{C} = \{C_1, \ldots, C_N\}$ and our network is end-to-end trainable via back-propagation.

4. Dataset

Since both of our feature extractors VGGNet and ResNet were pre-trained on ImageNet, we needed to use a different dataset to train/test our network on. This is important because we will need a different distribution in order to truly assess the performance on the unsupervised pre-

training task. To this end, we chose Caltech-256 Image Set [9].

Caltech-256 is a collection of 30608 images of 256 different categories, with more than 80 images in each category. In addition to offering a different distribution, its smaller size and fewer categories makes this dataset more manageable, and thus a better option. As a benchmark, we split the dataset into: 10249 for training, 1024 for validation, and 512 for testing.

Figure 3: Subset of Caltech-256 Images.

4.1. Data Preprocessing

We initially resized the images to:

$$[H * (64 + \epsilon)] \times [W * (64 + \epsilon)] \times 3$$

Then, divided evenly them into $H \times W$ cells. After that, we randomly sampled a 64×64 patch from each cell that we feed into our network. Such sampling avoids the network from over-fitting by just learning the edges.

In addition to resizing the image, we subtract the specified VGG Red, Green and Blue averages $(123.68, 116.78, 103.94)$ respectively from the image before passing it through VGG.

5. Experiments

5.1. Output Postprocessing

Our network described above could predict the same puzzle piece for multiple positions at test time. We know however that each piece appears only once in the correct reconstruction permutation. We therefore have to perform post-processing on the network output to fit this constraint of the problem. During the generation of the output sequence for each puzzle piece, we force our model to avoid predicting pieces that have already been assigned an index. We do this by setting the logits of already predicted or assigned puzzle pieces to be $-\infty$. This ensures that we do not predict that piece again.

5.2. Metrics

The primary metrics we used in assessing the performance of model on the Jigsaw Puzzle Task were:

1. **Loss:** At each time step in our prediction, we use Softmax loss on our model's prediction for that step. After predicting the full sequence f, we define the total loss as

$$L = \frac{1}{N+1} \sum_{i=0}^{N} L_i$$

where N is the number of pieces of the puzzle and

$$L_i = -\log\left(\frac{e^{f^i_{y_i}}}{\sum_j e^{f^i_j}}\right),$$

f^i_j is probability that the jth puzzle piece falls into the ith position in the correct reconstruction of the input. We augment our output space with an END token when predicting the sequence. We found that this helps the network converge better. Hence, there are $N + 1$ steps during prediction.

2. **Direct Accuracy:** Given the model's predicted sequence, S_m and the target sequence, S_t for a puzzle, we define direct accuracy as

$$A_d = \frac{1}{N+1} \sum_{i=0}^{N} \mathbf{1}[S_t(i) = S_m(i)].$$

This metric measures the ability of the model to predict correctly the absolute position of a puzzle piece. We can see from this that a model that randomly assigns a number $n \in \{0, 1, 2, \ldots N\}$ to each output sequence should have $\mathbb{E}[A_d] = \frac{1}{N+1}$. This metric was motivated by Sholomon *et al.*'s work in [20].

3. **Neighbor Accuracy:** Like the direct accuracy metric, this function was motivated by [20]. Formally, given the $2D$ representation of the model's solution g_s and target solution g_t, we have that

$$A_n = \gamma \cdot \sum_{i=0}^{H-2} \sum_{j=0}^{W-2} (\Gamma(i+1, j) + \Gamma(i, j+1))$$

where $\gamma = \frac{1}{2(W-1)(H-1)}$ and

$$\Gamma(i, j) = \mathbf{1}[g_t(i, j) = g_s(i, j)].$$

This metric measures the ability of our model to correctly predict the position of a piece relative to its neighbors in the the correct configuration.

5.3. Hyperparameter Tuning

The hyperparameters we chose to tune were the learning rate, number of RNN layers, size of the RNN and Jigsaw Fully Connected Dimension Size and optimization algorithm. In general, we found that higher batch sizes produced higher accuracies but we had to cap our batch size at 64 due to memory considerations.

5.3.1 Learning Rate

The first hyperparameter we tuned was the learning rate. We considered learning rates between $[10^{-6}, 1]$ and found the best learning rate to be roughly 10^{-4} with no significant difference in performance at small deltas around this learning rate.

5.3.2 Number of RNN Layers and RNN Cell

Once we fixed the learning rate, we considered the impact of the number of RNN layers on the performance of the model. We observed that a $2-$layer RNN produced the best results. We posit that the two layer network has a greater capacity for reasoning about the Jigsaw problem since chaining non-linear activations increases our network's expressive power. Increasing beyond this did not lead to any improvement and made training much more fragile. Additionally, we did not observe any significant improvement when using the LSTM Cell [11]. We defaulted to Gated Recurrent Unit (GRU) [3].

5.3.3 RNN and FC_Jigsaw Size

We performed grid search over RNN sizes within the range $[50, 1000]$ and sizes in the range $[128, 1024]$ for the output dimension of FC_Jigsaw (shown in green in Figure 2). We observed that, in general, the RNN size mattered more than the Fully Connected (FC) size. One possible explanation is that the RNN will have to reason about all N pieces whereas the FC layer only has to reason about each image piece. Hence, a higher capacity RNN will perform significantly better whereas a higher FC size has a marginal effect for higher sizes.

Figure 5 shows the dependence of the output dimension of FC_Jigsaw on direct accuracy of our validation set on a 2x2 puzzle. In general we observe a decreasing accuracy with increasing size. We hypothesize that the nature of the FC output size curve is due to the fact VGG and Resnet, are trained on significantly more classes (ImageNet 1000) [5] than our Caltech 256 [8] dataset. The embeddings produced thus contain features that might not be relevant to solving the Jigsaw problem on our dataset, and thus compressing the information from 2048 (VGG) / 8192 (Resnet) units to 256 allows our network to distill only the required information for inference.

Due to runtime and memory constraints, we could not go higher than RNN size of 3200. The results below describe our observation.

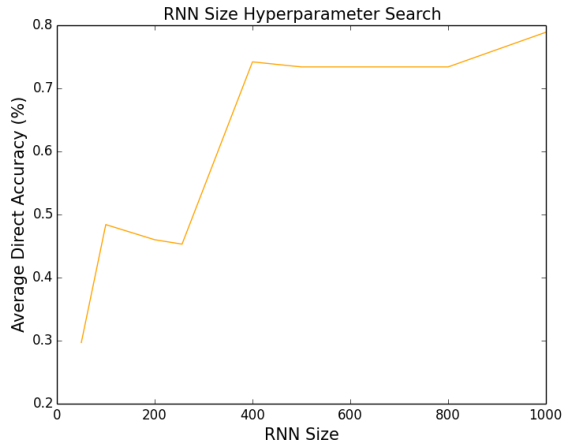

Figure 4: Search over RNN Size with fixed FC Dimension of 512.

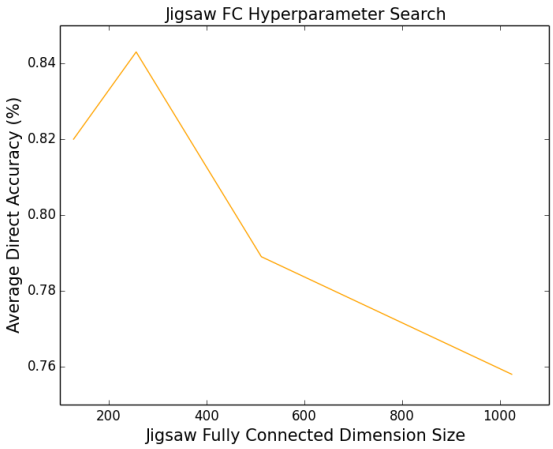

Figure 5: Search over FC Dimension with fixed RNN Size of 1000.

5.4. Performance on Different Puzzle Sizes

We evaluated our model on different Jigsaw Puzzle sizes. Figures 6 and 7 give Direct and Neighbor Accuracies on different sized puzzle tasks. In general, we find that we are able to perform significantly better than the random baseline for smaller puzzles but this gap drops as the puzzle size gets larger. Using Resnet as a feature extractor generally performed better than VGG. This is in agreement with results from a host of different Computer vision tasks [14].

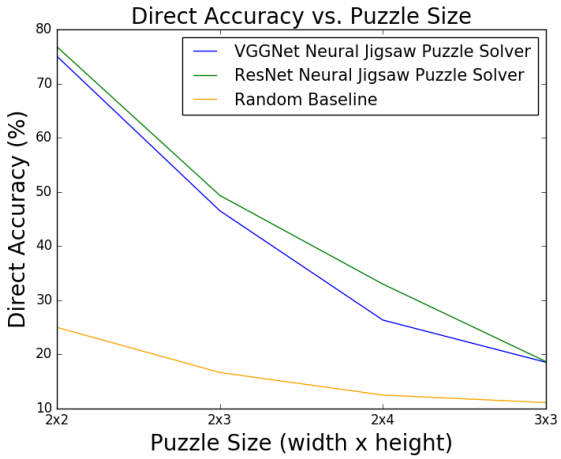

Figure 6: Direct Accuracies of Different Puzzle Sizes.

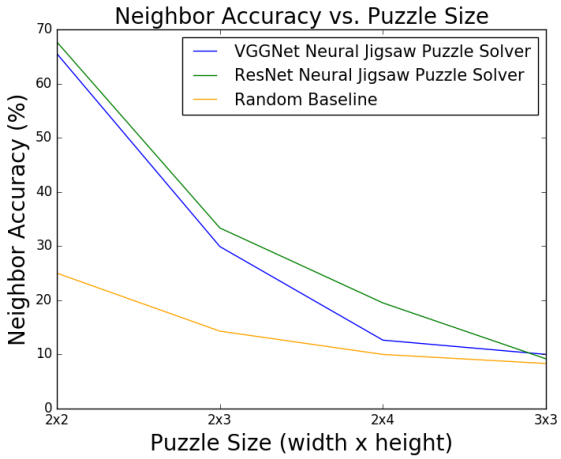

Figure 7: Neighbor Accuracies of Different Puzzle Sizes.

While the accuracy drops as the number of pieces increases, the rate at which it drops was not anticipated. We have a few hypotheses explaining why this is the case. One of them is that the nature of the problem has proved more difficult than expected for our network. Pointer Networks have traditionally been used to solve problems with lower dimensional input spaces - Convex Hull, Delauney Triangulation [26] in 2 dimensions. Thus, we suspect that scaling to dimensionality on the scale of this vision problem requires a more involved augmentation of the current architecture.

Furthermore, we conjecture that it would have been better to train a full model from scratch rather than use VGG or Resnet as a feature extractor. This is because either model is trained to detect certain features from each image piece and not necessarily features that describe the similarity between different pieces from the same image such as border similarity and relative feature positions. Doing this however, in the scope of this class would not have been feasible due to the large amounts of training data and time that would require to see appreciable results.

5.5. Saliency Maps

In order to understand how our model was interpreting the images, we created saliency maps of the original image using base code from the 2017 $CS231n$ Assignment 3 problem.

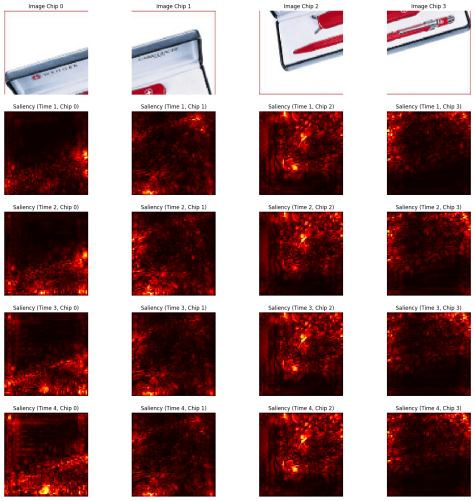

Figure 8: Grid of Saliency Maps on Trained Jigsaw Solver.

The saliency map shows how the saliency of each image piece changes as the model steps through time (down the grid) in determining the next index to assign to a position. Specifically, the first row after the image pieces shows the most important positions of each image used by the model in determining which index to assign next. We had hoped that our model would strictly focus on the contents of the image to make predictions. The maps however show that the model is focusing on both aspects of the image as well as the border.

5.6. Unsupervised Pre-training

Having achieved reasonable performance on solving the Jigsaw puzzle task, we sought to investigate whether our network had learned unsupervised representations that could be useful for solving the classification task on the Caltech 256 Dataset [8]. Taking inspiration from Noroozi and

Favoro [18], we built the two Neural Networks shown in Figure 9.

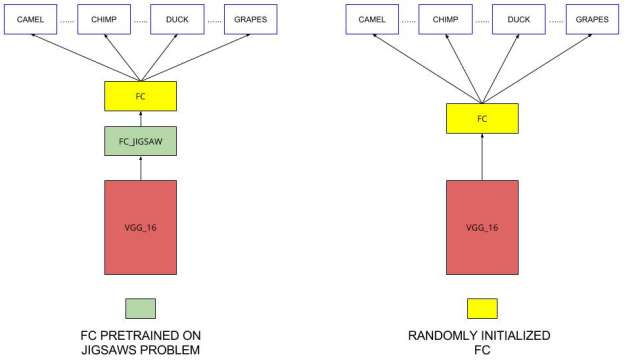

Figure 9: Neural Networks for image classification on Caltech 256 Dataset

The first network has a Fully Connected Network - (FC_Jigsaw) that is transfered from our 2x2 Jigsaw Puzzle Solver to this new task. On top of this is another Fully connected layer to output the class log-likelihoods. This fully connected is randomly initialized at the start of training. The second network has a single, randomly initialized MLP that transforms VGG [21] output features into class log-likelihoods.

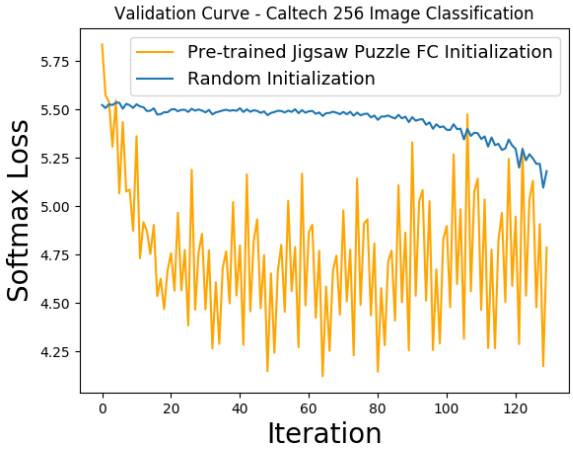

Figure 10

In training both Networks, we chose 40 classes out of the 256 possible classes in order to make the problem more tractable. Figure 10 shows the validation curve when both Networks were trained on 2560 images and validated on 1024 images. From this figure, we see that the Network that was augmented with FC_Jigsaw layer starts training in an already good position, with the validation loss for the second network only beginning to fall noticeably after 80 iterations. This supports our assertion that solving the Jigsaw Puzzle Problem allows us to learn semantically useful representations that can be transfered to other tasks like image classification. We are not immediately certain why the validation curve for the augmented network is much more unstable than that which was trained from scratch. We think it might have to do with the network having to adjust between the scales introduced by FC_Jigsaw and those of the actual classification problem. Smoothening out the curve would thus require a more involved search through the space of possible of initializations of top-most Fully Connected of the network.

6. Conclusion and Future Work

In conclusion, we have seen our best model is able to achieve 77% accuracy on the 2×2 puzzles using Resnet_50 as a feature extractor for the Pointer Network. We also presented some reasons explaining the sharp drop in direct and training accuracies as we increased the puzzle piece sizes. We argued that unlike previous problems tackled with Pointer Networks, this Jigsaw problem deals with embeddings in very high dimensions. Finally, we showed that our Jigsaw solver learned certain spatial properties inherent to the dataset. We demonstrated this by improving results on the classification problem by transferring the trained FC_Jigsaw layer unto that task.

Two possible extensions to this project are adding Reinforcement Learning [24] and Set2Seq [25] components. Reinforcement Learning allows us to use edge compatibility as a reward function. This edge compatibility function is similar to the Dissimilarity score defined in [20]. We could then train the network via policy gradient, effectively removing the need for labels. We expect that this will boost the performance of our model as it would better train it to prioritize edge similarities.

Additionally, Set2Seq allows us to disregard the order of our input and thus remove the probabilistic effect of input order on our output. [25] showed that the order of the input to Seq2Seq models has a considerable effect on the performance of the model and removing this constraints will most likely improve the performance of the model.

7. Acknowledgements

This project was built using Tensorflow [1]. We would also like to credit [13] for the Tensorflow Pointer Network implementation which we built upon.

References

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[3] J. Chung, C. Gulcehre, K. Cho, and Y. Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014.

[4] L. Davis. Handbook of genetic algorithms. 1991.

[5] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[6] H. EDELSBRUNNERi. Convex hull problem.

[7] D. Erhan, Y. Bengio, A. Courville, P.-A. Manzagol, P. Vincent, and S. Bengio. Why does unsupervised pre-training help deep learning? *Journal of Machine Learning Research*, 11(Feb):625–660, 2010.

[8] G. Griffin, A. Holub, and P. Perona. Caltech-256 object category dataset. 2007.

[9] G. H. Griffin and A. Perona. P. the caltech-256. *Caltech Technical Report, Tech. Rep.*, 2012.

[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. computer vision and pattern recognition (cvpr). In *2016 IEEE Conference on*, volume 5, page 6, 2015.

[11] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[12] K. L. Hoffman, M. Padberg, and G. Rinaldi. Traveling salesman problem. In *Encyclopedia of operations research and management science*, pages 1573–1578. Springer, 2013.

[13] Ikostrikov. Tensorflow pointer networks, May 2017.

[14] A. Jabri, A. Joulin, and L. van der Maaten. Revisiting visual question answering baselines. In *European Conference on Computer Vision*, pages 727–739. Springer, 2016.

[15] A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

[16] D. G. Luenberger. *Introduction to linear and nonlinear programming*, volume 28. Addison-Wesley Reading, MA, 1973.

[17] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 2015.

[18] M. Noroozi and P. Favaro. Unsupervised learning of visual representations by solving jigsaw puzzles. In *European Conference on Computer Vision*, pages 69–84. Springer, 2016.

[19] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv preprint arXiv:1511.06434*, 2015.

[20] D. Sholomon, O. David, and N. S. Netanyahu. A genetic algorithm-based solver for very large jigsaw puzzles. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2013.

[21] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[22] K. Son, J. Hays, and D. B. Cooper. Solving square jigsaw puzzles with loop constraints. In *European Conference on Computer Vision*, pages 32–46. Springer, 2014.

[23] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[24] R. S. Sutton and A. G. Barto. *Reinforcement learning: An introduction*, volume 1. MIT press Cambridge, 1998.

[25] O. Vinyals, S. Bengio, and M. Kudlur. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*, 2015.

[26] O. Vinyals, M. Fortunato, and N. Jaitly. Pointer networks. In *Advances in Neural Information Processing Systems*, pages 2692–2700, 2015.

[27] K. Xu, J. Ba, R. Kiros, K. Cho, A. Courville, R. Salakhudinov, R. Zemel, and Y. Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International Conference on Machine Learning*, pages 2048–2057, 2015.

[28] R. Yu, C. Russell, and L. Agapito. Solving jigsaw puzzles with linear programming. *arXiv preprint arXiv:1511.04472*, 2015.