Jigsaw Puzzle Solving: Deep Learning Approaches

[CS503 ML Course Project Report, Fall 2018]

Subhranil Bagchi[*]
M.S.(R.)
2018csy0002@iitrpr.ac.in

Anurag Banerjee[†]
M.Tech.
2018csm1007@iitrpr.ac.in

Anurag Singh Mehta[‡]
B.Tech.
2016csb1111@iitrpr.ac.in

ABSTRACT

This report discusses some Machine Learning approaches incorporated to solve the Image Jigsaw Puzzle problem. The puzzle can be perceived as a classification problem (1.1). The methodologies used include CNN, LSTM and Autoencoder. They have been compared for the same dataset and in case of CNN also evaluated on different dataset. Varying degree of accuracy was obtained in each case which depended heavily upon dataset size, learning rate and other hyper-parameters. A discussion on the implementation and the results have been included.

Keywords

Jigsaw Puzzle; CNN; LSTM; Autoencoder; Hyper-parameter; tensorflow; keras

1. INTRODUCTION

This section will first describe the Jigsaw puzzle/problem and then discuss the complexity involved. It will also mention the motivation behind solving such a problem, in brief.

1.1 What is a Jigsaw Puzzle

In an image jigsaw puzzle, an image is cut into several pieces,and the pieces are presented to a player in some randomly jumbled order. The challenge for the player is to either discover the given order or rearrange the pieces in correct order so that the original image may be reconstructed.

From the perspective of Machine Learning, the puzzle can be formulated as a classification problem; if the image is is cut into **n** pieces, then there are **n!** possible orderings of the pieces. Thus, there are $n!$ classes and the ML algorithm attempting to solve the puzzle has to predict the class (the sequence) for given order of image pieces.

[*]CNN Approach
[†]LSTM Approach
[‡]Auto-encoder Approach

CS503 ML November 26, 2018, IIT Ropar
ACM ISBN .
DOI:

1.2 Why is it a hard problem

Solving a Jigsaw Puzzle is a known NP-Hard Problem. For a single image, if it is broken into n parts, it can have one of the $n!$ possible permutations as discussed above. Solving such a hard problem is extremely non-trivial. Till date no such algorithm exists that can solve the named problem efficiently (with 100% accuracy). For a small number it is very easy, but as pieces goes higher, for example, take only 16 pieces for an image, it would result in 16! permutations. Comparing each pieces with every other piece to form the actual image is extremely time consuming even for a powerful machine. Dynamic programming can be applied to make the computation somewhat faster, still it is not efficient.

That being said, deep learning is a powerful tool to discover pattern in a given data or to classify data. Through deep learning techniques one can hope to learn the cuts in an image from the pieces. The Jigsaw puzzle problem can result in exponential number of classes even for a small number of pieces in an image. This makes it such a big challenge, and our goal is evaluate it's performance of the stated problem. We however limit our goal to the case of axis aligned cuts, wherein we cut an image in 4 equal parts.

2. RELATED WORK

Jigsaw puzzle has multiple real life application, some of them being – merging shredded data, in cryptography, etc. It still remains such a hard problem to solve and one needs to determine complex structures amongst the pieces of the image and localization of objects so that they form meaningful semantics. Before machine learning, most of the works in this field were closely related to graph theory. [4] suggests a probabilistic approach where they treat every object of interest as a node and based on that they calculate the contribution of each node to multiple graphs. They then calculate the dissimilarity measure among the nodes in the graphs formed and the error is measured in terms of sum of squared coloured difference along the boundary. Finally one optimal graph is selected.

The problems with these kinds of approaches is that they are extremely subjective, and may work well, or not well, depending on the images they are applied too. Machine Learning on the other hand tries to gauge the patterns that can find the location of each piece in the image. Some recent work in this field from the perspective of Machine Learning include [7], [5] and [8]. In [7], they try to solve the puzzle using multiple machine learning approaches, including CNN, LSTM, Random Markov Fields and Multi-Layered perceptron, and compare accuracies of the results. They try it for two types of features, one for raw image and the other one, Alexnet Features. We will closely follow their approach. [3] uses an unsupervised training, post that it applies the pre-processed data to the CNN and RNN model. [4] follows the representation learning approach and context free networks which made us extend our project to the use of autoencoders in solving the puzzle.

3. METHODOLOGY

In this we will first describe the Dataset that we use then discuss why classical machine learning approaches will not work. Then each of the method used will be described in brief along with our experimental setup.

3.1 Dataset

The Caltech-101 [6] was used for our experiments. The entire dataset was divided into 4:1 ratio of training and testing. The results of different methods were compared based on this dataset.

Additionally, another dataset was used [1], specifically for the CNN model, apart from the Caltech-101, to check it's general consistency. This dataset consisted of the Validation Set of Open Images and Images from CVCL MIT. These images consisted of natural images (how daily life images should be); the images were split similar to the Caltech-101.

3.2 Classical ML Techniques

Jigsaw puzzle solver stands out to be an extremely difficult task to learn because to classify the sequence of image pieces correctly, the model needs to learn the weights/parameters that can find position of a given piece when compared to all the other pieces, and not just any distinctive feature in the image. This tends to become an extremely hard task and the classic machine learning models (such as a logistic regression based classifier or even an SVM) tend to fail (as is proven in [2]). The major factor being, these models do not consider context - one piece is not independent from other pieces.

Thus a deep learning approach is preferred to solve the problem. For our purpose we have chosen three approaches – first is a CNN model that takes the pieces of the image combined together as an image; the second approach is an LSTM based one, where the pieces are given as raw inputs as a sequence series; and finally an autoencoder model where the input is similar to CNN model but the output is the concatenation of the pieces in the correct order (the correct image as a whole); here learning is achieved using deconvolution and fractional strides.

In the following subsections, we will describe the methods and setups that we have used to implement them.

3.3 The CNN based Model

Convolutional Neural Networks (see Fig. 1) are extremely powerful kind of neural network technique when it comes to images. They required minimal preprocessing and images are generally directly given as input to the neural network The CNN basically consists of the convolutional layer and the pooling layer, followed by a few number of dense layers. CNN are know to perform convolutions of very image, (in mathematical term more of cross-correlation, rather than convolution).

3.3.1 Experimental Setup

Pre-processing
The Images were preprocess into 128×128 grayscale images. These images were cut into 4 parts from the centre, and were rearranged in different order over the same image. These new images were feeded as raw input to the first convolutional layer. The same preprocessed imageset was used for auto-encoder model. **CNN**
The architecture of the CNN model comprised of 6 Convolutional + Pooling layers, 3 densed (fully connected layer) and one output layer. Extra-layers were added for hyper parameter tuning. The batch size was taken to be 96 for both the training and testing set (other batch sizes were also taken into account for hyper-parameter tuning). The Adam Optimizer was used, learning was done on the basis on cross-entropy loss function. The code for this model is in tensorflow and is developed based on a CNN implementation available at [3]. The filter size was taken to be 5×5, with number of filters for 6 layers being 13, 32, 48, 64, 96, 128, in increasing order of depth. To offer regularization to the training model, the pieces order for each images were shuffled after every 3 epochs.

3.4 The LSTM based model

Recurrent Neural Networks are NNs with a feedback loop. This provides them with ability to remember the context (see Fig. 2) or the information of previous iteration. As shown in the figure, we may consider the LSTM cell as a chain of cells.

Due to this inherent chain structure, an LSTM is best suited to learn from time series and sequential data. This standard RNN, however, faces the problem of

- Vanishing Gradient
- Exploding Gradient

since the results are constantly multiplied (*vector multiplication of the learned features and weights*), the values can either increase drastically or reduce to zero. In effect, the RNN fails to maintain the context anymore. This problem is solved by RNN-LSTM.
A **Long Short Term Memory** or LSTM is a special type of RNN which is used to overcome the above mentioned problems. An LSTM cell (see Fig. 3) has the means to select how much of previous information is to be retained. The output from previous cell and the input at current time step is used to decide the value of cell memory from previous to present cell apart from calculating the output for current cell and adding to the cell memory. The ability to forget helps the LSTM to avoid build-up of long chain of unnecessary context and thus prevents the problems of a standard RNN.

3.4.1 Experimental Setup

The LSTM network was trained and tested on the CALTECH - 101 image dataset which consists of a total of 9145 images. The images were first resized to 128×128 px. Each image was cut into four equal parts of dimension 64×64 px and stored in matrix of dimensions 36580×4096. This image matrix was normalised and set was split into training and testing set by the 80:20 rule. All the possible sequences/orderings were generated for the parts, resulting in a matrix of dimensions $175200 \times 4 \times 4096$. All 4! permuta-

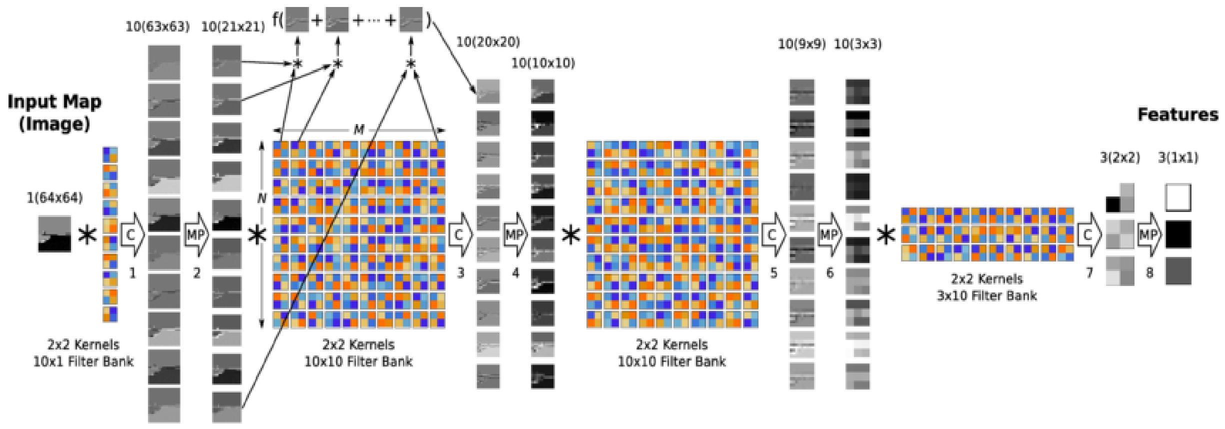

Input Map (Image)
1(64x64)
2x2 Kernels 10x1 Filter Bank
10(63x63)
10(21x21)
2x2 Kernels 10x10 Filter Bank
10(20x20)
10(10x10)
2x2 Kernels 10x10 Filter Bank
10(9x9)
10(3x3)
2x2 Kernels 3x10 Filter Bank
Features
3(2x2)
3(1x1)
M
N
C
MP

Figure 1: CNN Model

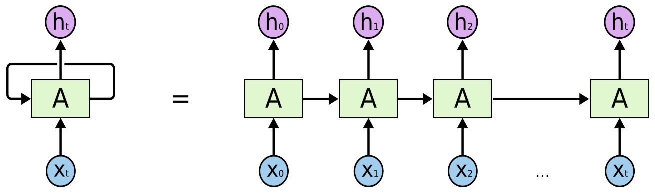


Figure 2: An unrolled RNN

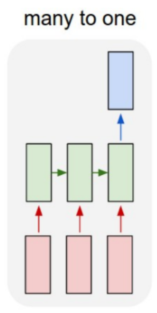


Figure 4: LSTM many-to-one model

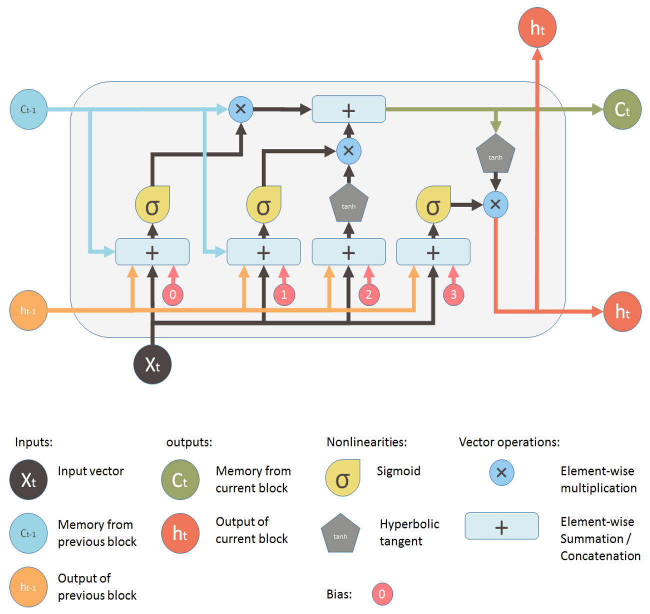


Figure 3: LSTM Cell

tions were generated for each image to ensure that the system can learn how cuts are made in an image – essentially it has to learn the edges where cuts can be made in an image.

The LSTM network itself was setup using Tensorflow, based on the implementation available at [2]. This is a many-to-one model (see Fig. 4). Various hyper-parameters for the LSTM network with three stacked cells were:

- batch-size
- learning rate
- cell forget bias
- number of training iterations
- number of hidden units in LSTM cell

These were tuned (as can be seen in the Results section) to get an accuracy for classification of around 74-75%

3.5 Auto-encoder

Autoencoding is a data compression algorithm where the compression and decompression functions are

- data-specific
- lossy, and
- learned automatically from examples rather than engineered by a human

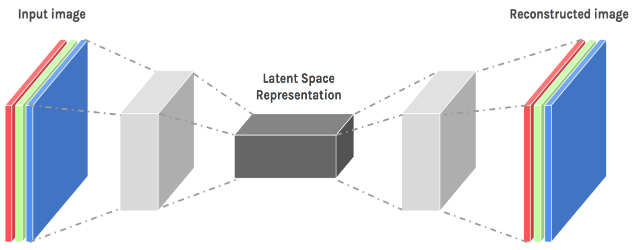


Figure 5: Diagrammatic representation for an Auto-encoder

Additionally, in almost all contexts where the term *autoencoder* is used, the compression and decompression functions are implemented with neural networks.

The number of classes among which the pattern needs to be classified are $n!$. This makes it intractably complex to classify if the number of pieces is very high. Thus, we have used the auto-encoder, as this could solve such a problem. An auto-encoder can encode and regenerate the same image. This mechanism, we have taken to the next level. For our purposes we have supplied it with jumbled images and auto-encoder predicts a lossy image. We then compare the cut-portions of the image to the original image, and assign the position value. This position value is compared to the target position. This takes O(n^2) time complexity solve the problem.

3.5.1 Experimental Setup

To build an autoencoder, we need three things: an encoding function, a decoding function, and a distance function between the amount of information loss between the compressed representation of the data and the decompressed representation (i.e. a "loss" function). The encoder and decoder will be chosen to be parametric functions (typically neural networks), and to be differentiable with respect to the distance function, so the parameters of the encoding/decoding functions can be optimized to minimize the reconstruction loss, using Stochastic Gradient Descent.
Two interesting practical applications of autoencoders are data denoising and dimensionality reduction for data visualization. With appropriate dimensionality and sparsity constraints, autoencoders can learn data projections that are more interesting than PCA or other basic techniques.

We have implemented convolutional autoencoder. Since our inputs are images, it makes sense to use convolutional neural networks (convnets) as encoders and decoders. In practical settings, autoencoders applied to images are always convolutional autoencoders – they simply perform much better.

The encoder will consist in a stack of Conv2D and MaxPooling2D layers (max pooling being used for spatial down-sampling), while the decoder will consist in a stack of Conv2D and UpSampling2D layers. The images are first cut into 4 pieces each, then shuffled in the uniformly such that each of the 24 different possible cases get uniform number of images and then fed as input to the system. The target class is the default order of the four pieces of each image i.e., the original image itself.

4. RESULTS

Table 1: CNN Results

Dataset	Layers	Learning Rate	Batch Size	Accuracy
Open Image	6 CL, 3 D	1e-4	96	86.871%
Open Image	7 CL, 4 D	1e-6	128	84.12%
Caltech-101	7 CL, 4 D	1e-4	32	72.628%

Table 2: LSTM Results

Learning Rate	Batch Size	Accuracy
0.000548	256	75.9%
0.000448	128	74.5%
0.000648	128	71.6%
0.000548	64	69.1%

In this section we shall summarize the results obtained for each of the methods used. We also present a table (see Table **??**) for making a quick comparison.

4.1 Result for CNN Model

The learning rate was kept to be equal to 1e-4, in general as this produced the optimal result, however, for tuning purposes the value was changed to 1e-6 and 1e-5. The batch size was taken to be 96, but, was also tested for the values of 128 and 32. The weights were truncated normal.

The results are summarized in Table 1. The run results can be seen in Figures 7, 8.

4.2 Result for LSTM Model

The stacked LSTM cell (3 in number) have the following fixed parameters:

- LSTM cell 1 forget bias: 1.4
- LSTM cell 2 forget bias: 2.0
- LSTM cell 3 forget bias: 1.9
- Number of units in hidden layers: 256

The results for various hyper-parameter tuning may be seen in the Table 2. The results of the best run are shown in Figures 9, 10.

4.3 Result for Auto-encoder Model

The model was set to have 4 convolution layers (with 128, 128, 64, 64 filers of size 3×3 for each layer respectively) and 4 deconvolution layers (with 64, 64, 128, 128 filers of size 3×3 for each layer respectively) The accuracy was found to be around 20%. The autoencoder produces the output as shown in Figure 11

5. SUMMARY AND FUTURE WORK

In this project, we have attempted to determine how well the currently known Deep Learning models are fit to solve the Jigsaw puzzle problem. Subject to hyper-parameter tuning, availability of dataset and hardware resources available, it was observed that Deep Learning methods were able to solve the puzzles around 70% of the time. From the results we see that amongst the different techniques we used, the LSTM model tends to provide the best results on the same Caltech-101 dataset. CNN fits to be second in line, but it generalized well (good results independent of the training

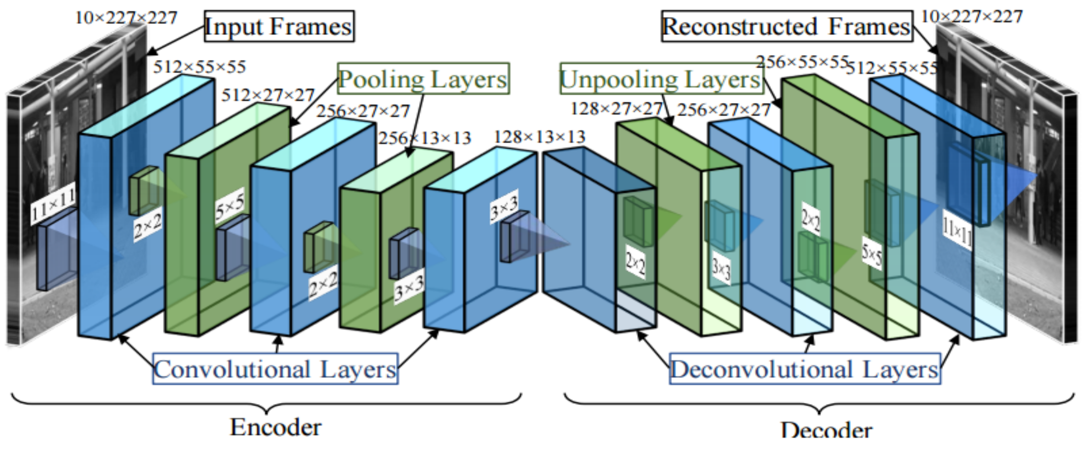


Figure 6: Encoding-Decoding process for an Auto-encoder

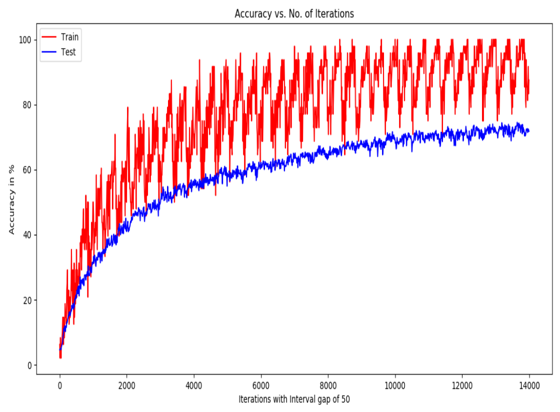


Figure 7: CNN Caltech-101 result

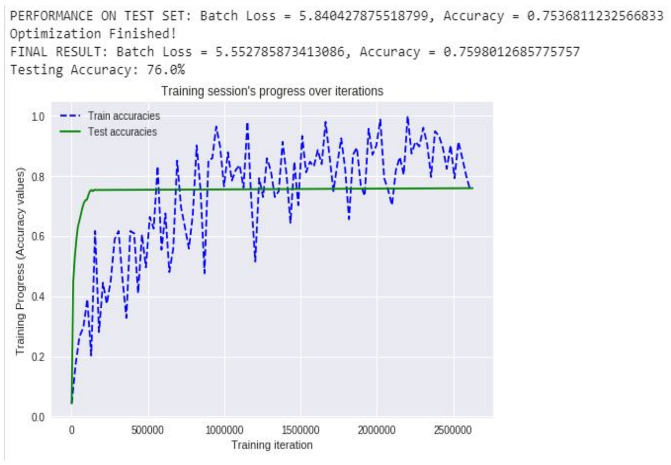
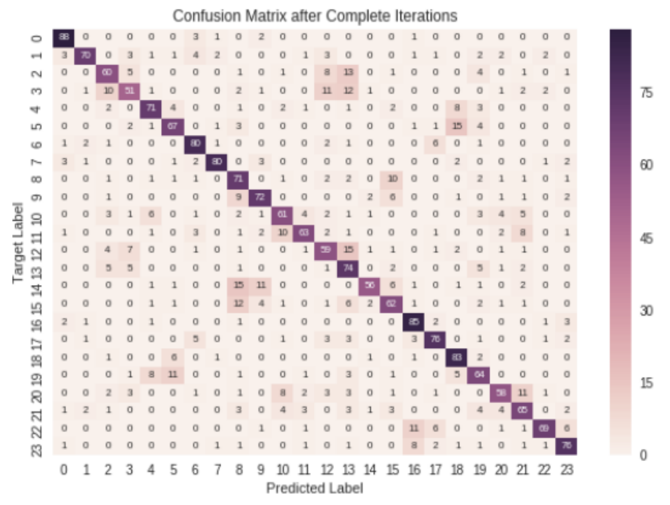

```
PERFORMANCE ON TEST SET: Batch Loss = 5.840427875518799, Accuracy = 0.753681123256683
Optimization Finished!
FINAL RESULT: Batch Loss = 5.552785873413086, Accuracy = 0.7598012685775757
Testing Accuracy: 76.0%
```

Figure 9: LSTM Caltech-101 result



```
Final Tesing Accuracy: 72.628%
```

Figure 8: CNN Confusion Matrix

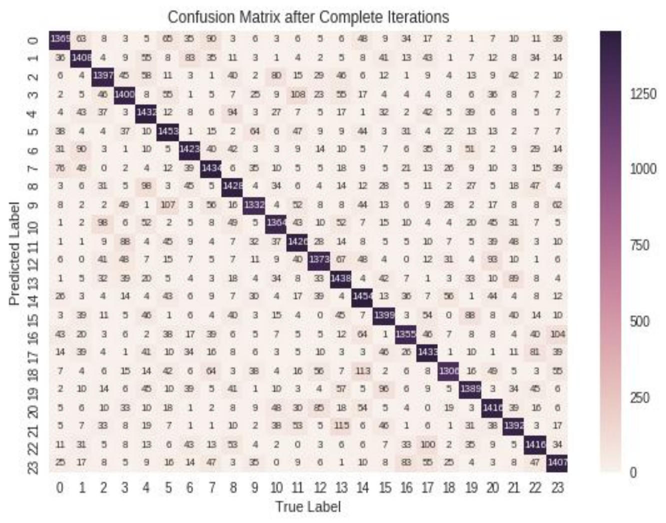


Figure 10: LSTM Confusion Matrix

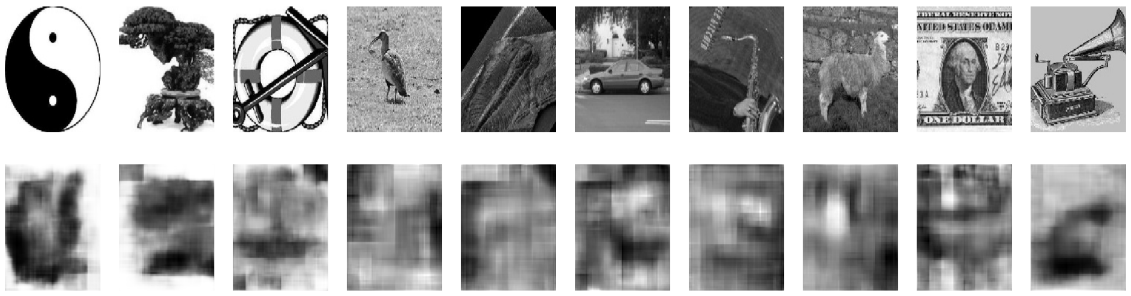


Figure 11: Auto Encoder Output

set used) and autoencoder models to solve a problem like this is still in its infancy (poor result).

In the future, given the high accuracy of both CNN and LSTM, we would like to combine them into a single model and evaluate the performance. Also, we plan to extend our work to solve images with much larger number of pieces, for example 16 pieces, for which we would like to implement more fine tuned version of autoencoders, as solving for such large number of pieces using the traditional models will result in exponentially large number of classes, and hardware limitations shall pose a serious challenge, even for GPUs.

6. ACKNOWLEDGMENTS

We are extremely grateful to our course instructor, Dr. Narayanan C Krishnan for his constant guidance and support. We would also like to thank google for the amazing service for cloud based GPU. We also wish to thank our colleagues for valuable inputs and support.

7. REFERENCES

[1] https://github.com/cvdfoundation/open-images-dataset#download-full-dataset-with-google-storage-transfer. Accessed: 2018-11-26.

[2] G. Chevalier. LSTMs for human activity recognition. https://github.com/guillaume-chevalier/LSTM-Human-Activity-Recognition, 2016. Accessed: 2018-11-26.

[3] G. Chevalier. https://github.com/Hvass-Labs/TensorFlow-Tutorials, 2017. Accessed: 2018-11-26.

[4] T. S. Cho, S. Avidan, and W. T. Freeman. A probabilistic image jigsaw puzzle solver. 2010.

[5] L. Dery, R. Mengistu, and O. Awe. Neural combinatorial optimization for solving jigsaw puzzles: A step towards unsupervised pre-training.

[6] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental bayesian approach tested on 101 object categories. *Computer vision and Image understanding*, 106(1):59–70, 2007.

[7] V. Kulharia, A. Ghosh, N. Patil, and P. Rai. Neural perspective to jigsaw puzzle solving.

[8] 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.