COL870: Assignment 2

Kartikeya Badola and Suraj Joshi

2018EE10221

2018MT10045

ABSTRACT

This report contains the details of the experiments performed in Assignment 2 of the course COL870. The objective of this assignment is to build a sudoku-solver using Conditional GAN and Recurrent Relational Network. For part 1 of the assignment, we generate a labelled dataset using the Sudoku images provided and train a Conditional GAN on it. For part 2 of the assignment we implement a Recurrent Relational Network that serves as a sudoku solver giving symbolic outputs. In part 3, we improve the performance of both these models on the dataset by improving the clusters made. We provide details of implementations of these models and experiments that we have done and report our observations. We use PyTorch for our implementations.

Keywords: Deep Learning, Generative Adversarial Network (GAN), conditional GAN (CGAN), Recurrent Relational Network (RRN), Sudoku-solver

1 PART 1 AND 2

Intention of this assignment was to build a visual sudoku solver with only 9 labels given (1 blank and 8 digit classes). The dataset has 10,000 sudokus and roughly 50% of digits in the sudoku are blank. We can extract 640,000 images for the 8 classes and need to assign a label to each one of them.

Using K-Means for such a task is not a good solution due to the following reasons:

1. K-Means doesn't take into account the structure inherent to images and just treats them as a linear array. This is a poor choice of representing images.

2. K-Means works on euclidean distance and due to curse of dimensionality, it is not a good metric to optimize.

3. K-Means is unsupersvised and hence there isn't any method to make sure that each of the 9 labelled images fall into different clusters.

Dimensionality reduction using PCA will not work since the components having the highest variance in images are due to lighting variability.

In SigNet, the Siamese neural network paper [1], 1-neighbour classifier on images performs very poorly on the one-shot task suggesting that modifying K-Means to take into account the 8 labelled images will also not work.

We know from studying CNNs that images have spatial locality implying that the individual pixels which are close together in an image are likely to be strongly interdependent. These dependencies are ignored in other techniques like MSE or PSNR which use absolute errors. They are important because they carry important information about the larger structure that forms the complete scene represented in the image. **Structural Similarity (SSIM)** [2] is a perception-based model that considers image degradation as perceived change in structural information, while also incorporating important perceptual phenomena, including both luminance masking and contrast masking terms.

SSIM index is a way to measure the similarity between two images. It can be viewed as a quality measure of one of the images being compared, provided the other image is regarded as of perfect quality.

The SSIM index is calculated on various windows of an image. The measure between two windows x and y of common size N×N is

SSIM Index formula: $\frac{(2\mu_x\mu_y+c_1)(2\sigma_{xy}+c_2)}{(\mu_x^2+\mu_y^2+c_1)(\sigma_x^2+\sigma_y^2+c_2)}$

where

μ_x, μ_y are the means of x and y respectively

σ_x, σ_y are the standard deviations of x and y respectively

σ_{xy} is the covariance of x and y.

$c_1 = (k_1 L)^2, c_2 = (k_2 L)^2$ are two variables that stabilize division with the weak denominator, where usually $k_1 = 0.01, k_2 = 0.03$ and L is the dynamic range of the pixel values.

We use pytorch_msssim library to calculate SSIM index.

Using SSIM as metric, we can measure the distance between an unlabelled image and a labelled image. Since we don't need to partition the entire dataset (GAN doesn't need 64000 images to be trained), choosing only those images which are similar to some labelled image with confidence more than a certain percentage improves the quality of clusters a lot. This was extremely beneficial for our model because of which our CGAN was able to generate images from all classes without any supervision.

1.1 Conditional GAN

1.1.1 Dataset Generation

We generate GAN dataset using SSIM. We compute SSIM for each unlabelled image with the ground truth to generate 8 scores. The class selected is the one which has the highest SSIM score and contributes to atleast 30% of all 8 SSIM scores. We retain roughly 400,000 images using this method.

Once we collected the images we observed an imbalance in the number of digit samples available for each class. To counter this imbalance, we fixed each class to utilize only the minimum number of samples across all classes (around 40000).

These classwise clusters of query data also allow us to run classwise FID later on. This also gives us the classwise spread of images in all the sudokus in the train and query sets.

1.1.2 Architecture and Training

We follow the same implementation from [3]. The model presented in this paper is extremely weak because of which the authors have to train it for a really large number of epochs. Since this was not feasible to be run on colab, we made just two small changes in the generator (which added negligible number of parameters). Rest all hyperparameters are exactly the same as given in the paper. We were inspired from the following repository [4]

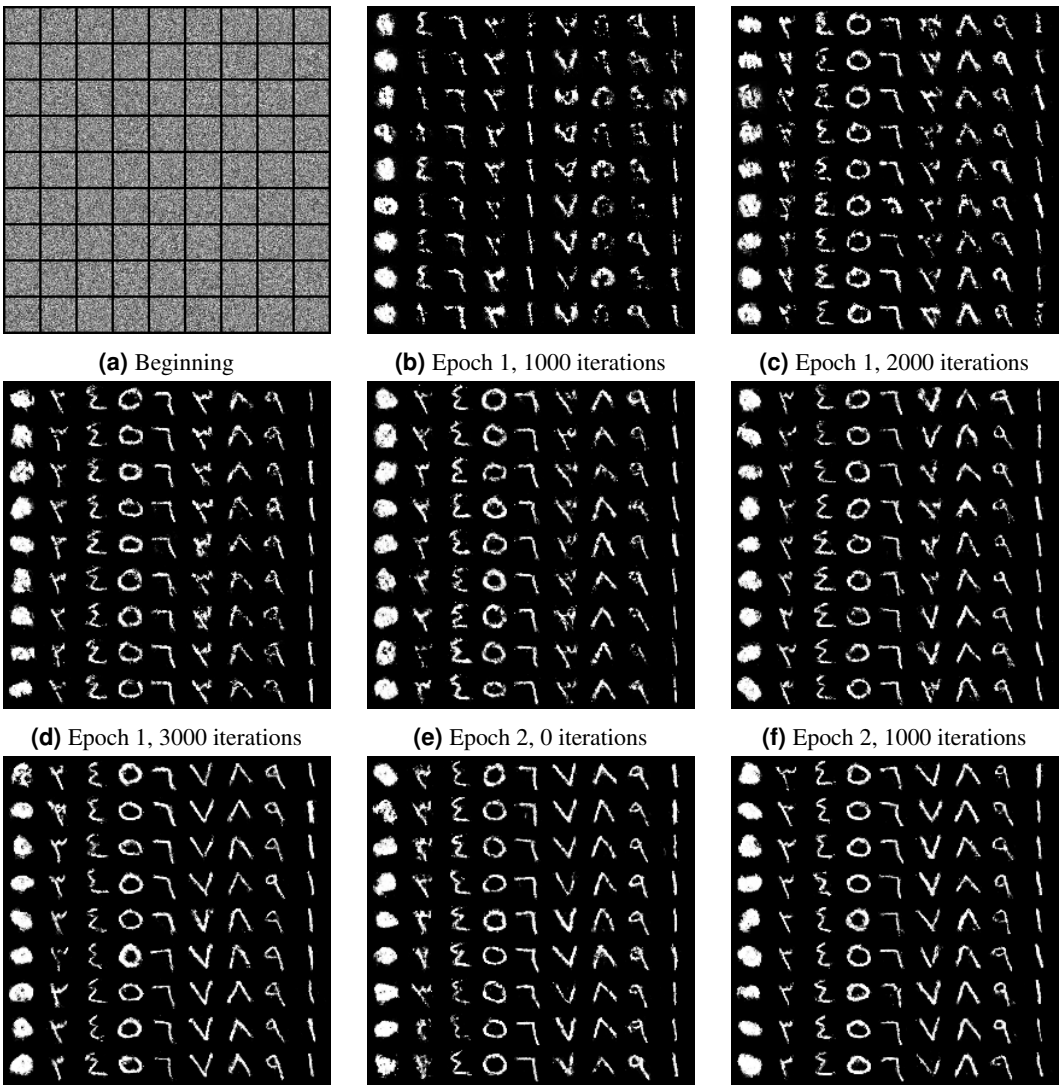
1. Replaced ReLU with LeakyReLU

2. Added BatchNorm in z and zy net (only in two layers)

The discriminator takes as input an the output of the generator and the same labels. Discriminator is exactly the same as proposed in the original paper.

For both the generator and the discriminator, SGD optimizer with learning rate = 0.1 and momentum = 0.5 were used with ExponentialLR scheduler.

This model was trained for 5 epochs, using the digit generated from SSIM. As opposed to K-Means clusters, we noticed that SSIM improves the diversity in the training set and allowed the GAN to learn to generate better images. Some samples of images from each of the 9 classes that the GAN was learning are as follows. The GAN learnt the best representation by the end of the second epoch (the final image) although we finished the training till the end of 5 epochs and used the best model for further parts.



(a) Beginning **(b)** Epoch 1, 1000 iterations **(c)** Epoch 1, 2000 iterations

(d) Epoch 1, 3000 iterations **(e)** Epoch 2, 0 iterations **(f)** Epoch 2, 1000 iterations

(g) Epoch 2, 2000 iterations **(h)** Epoch 2, 3000 iterations **(i)** Epoch 3, 0 iterations (best model)

Figure 1. Learning to Generate Digits

1.1.3 Evaluation

The generator so trained was then used to create 9000 images, 1000 per class, which was stored in gen9k.py as per the instructions. The 9000 target images for the same were then stored in target9k.py.

We also tested the quality of images generated by the GAN we trained by evaluating FID score on 5000 generated (by the best model isolated from the training) samples of each class. We found the classwise FID scores as well as the overall FID score. These are reported in the following table.

Class	0	1	2	3	4	5	6	7	8
FID	101	120	99	122	86	77	119	99	70

1.2 Image Classifier

We generate a classification dataset (for classes 0-8) using the CGAN which we trained in the last step. We use LeNet classifier to learn on these images, we choose this because we do not want to overfit on the dataset created by the GAN. This LeNet is trained with Cross Entropy Loss with Adam Optimizer for 5 epochs. We obtain 100% accuracy on the validation set.

This classifier quickly learnt to transform the digits into the symbolic representations (integers) for the RRN.

1.3 Recurrent Relational Network

1.3.1 Architecture and Training

The architecture for the RRN closely follows the paper implementation [5]. The model takes as input a pair of sudokus, one which is to be answered and one which contains the correct solution.

It first creates the embeddings for the row, column and sudoku input. The dimensions of all these embeddings is 16.

These three embeddings are concatenated and input into a multi-layered perceptron (MLP) which has 3 linear-ReLU layers with a final linear layer, and this output gives the fixed node feature vector. Each MLP produces a 96 dimensional output.

The message passing takes care of the row, columns and box constraints of sudoku by using a mask and only using those messages that are allowed for a given cell in the sudoku.

```
tensor([[0., 0., 1., 0., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0.],
        [1., 1., 1., 1., 0., 0., 0., 0.],
        [1., 1., 0., 1., 1., 1., 1., 1.],
        [0., 0., 1., 0., 0., 0., 0., 0.],
        [0., 0., 1., 0., 0., 0., 0., 0.]])
```

(a) Mask for Patch at (6,3)

Figure 2. Example of Sudoku Mask for Message Passing

Then following the paper, the messages are passed and the concatenation of the earlier output and the constantly changing messages are passed through another MLP (with the same architecture). The output of this MLP is passed as the LSTM cell input with the earlier node feature vector. The cell state is initialised as zeros according to the batch size and is implicitly updated. The LSTM cell used has a hidden dimension of 16. The output hidden state of the LSTM is passed through a linear layer with 9 outputs to get the logits.

This is repeated 32 times for each batch for a single step during the training according to the paper.

Since the loss is taken per step, we sum the Cross Entropy Losses (with the answer sudoku) of the LSTM output using the output logits as obtained earlier across all 32 iterations, and this is used as the total loss. Moreover, the last output from the 32 iterations is returned as the final output for prediction. Taking the max of the logits returns the prediction for the symbolic representation of a given input symbol.

1.3.2 Evaluation

We provide sudoku representations as input and calculate both the individual digit accuracy as well as the sudoku accuracy i.e. if the entire output matches the answer sudoku implying it has solved the sudoku correctly without violating any constraints.

This model is trained for 100 epochs and the output is the statistics of this training are as follows. We use validation loss to find the best model which we find at the end of the 95th epoch.

(a) Losses **(b)** Sudoku Accuracies **(c)** Digit Accuracies

Best Model	Loss	Sudoku Acc	Digit Acc
Train	0.6367	0.0007	0.8120
Val	0.6519	0.0025	0.8010
Test	0.6586	0.0000	0.8009

Figure 3. Evaluation Metrics for RRN

As we can see from the images, the digit accuracy steadily improves to around 80% at the end of the training. The sudoku accuracy doesn't improve much from 0 which is expected given that it is unlikely for all 64 predictions to be correct with our digit accuracy of 80%.

2 PART 3

We try to improve on the results obtained in part 1 by changing our approach from an unsupervised approach to a semi-supervised approach, given that we have the true digit/sudoku solutions available to us that we can utilize for better performance on the task.

2.1 Siamese Networks

This problem should be best thought of to be similar to 'signature verification' problem where the database of signatures is huge but examples in each class is limited. In such cases, Siamese Neural Networks [6] is the best way to go as it tries to maximize the distance between representations of images from two different classes. Siamese neural network is a class of network architectures that contains two identical subnetworks. The twin CNNs have the same configuration with the same parameters and shared weights. The parameter updating is mirrored across both the subnetworks. These subnetworks are joined by a loss function at the top, which computes a similarity metric involving the Euclidean distance between the feature representation on each side of the Siamese network. One such loss function that is mostly used in Siamese network is the Triplet Margin Loss.

2.2 Invariant Information Clustering

Invariant Information Clustering: IIC is an unsupervised clustering objective that trains neural networks into image classifiers and segmenters without labels, with state-of-the-art semantic accuracy. The method and operates on any paired dataset samples. The objective is to maximise mutual information between the class assignments of each pair. IIC is robust to some pitfalls of other clustering algorithms, like degenerate solutions, and noisy images in the dataset which it deals with by auxiliary overclustering. [7]

Sudoku dataset has a structure in the sense that images present in the same row or column cannot belong to same class. This can help us to generate the negative pairs required in Siamese Neural Networks. However we still need examples of positive pairs. We take inspiration from IIC and claim that an image and its augmentation (using affine transform) will belong to same class.

This enables us to extract perfect clusters with NO constraint violations. Our CGAN and RRN greatly benefit from such a dataset.

We additionaly train a classifier which distinguishes 0 from other classes. SSIM correctly classifies 0 classes so such a dataset is easy to build from previous pipeline.

2.3 Siamese Architecture

We use a ResNet for classifying sudoku patches to their proper numerical counterparts. The ResNet we use is from our Assignment 1 implementation with $n = 4$ residual blocks. The training method we use

involves creating positive and negative examples and then using triplet loss during training. We prepare the DataSet such that it returns the triplet of an anchor (the image itself), an affine transform of the image (positive image) and a patch from a different digit (negative image).

During training, all three, the anchor, positive and negative examples are passed through the model sequentially and create 64 dimensional embeddings in the penultimate layer (explaining absence of FC layer in our model) and the outputs are used in calculating the Triplet Margin Loss. We use a margin of 2.0 for the training.

2.4 Conditional GAN

We use the same architecture as in part 1 for the CGAN.

The CGAN is also trained in the same way as in part 1. This time, we obtained the best model at 7 epochs.

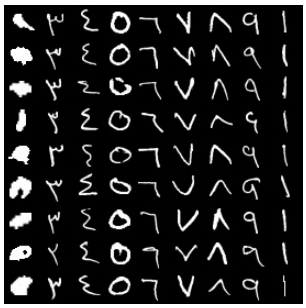


Figure 4. Images produced by the best generator

2.4.1 Evaluation

The FID Scores obtained from the final CGAN are as follows:

Class	0	1	2	3	4	5	6	7	8
FID	80	67	65	80	44	60	56	98	45

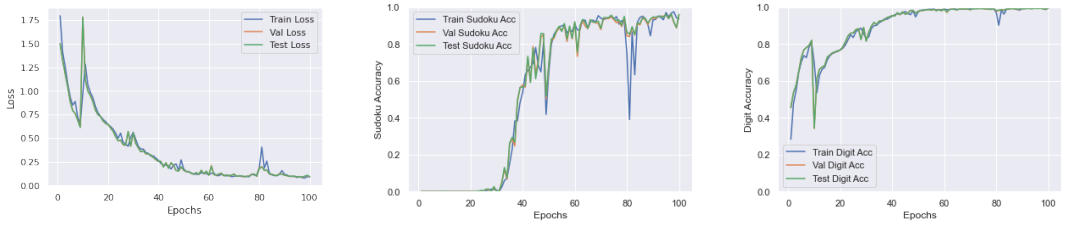
Observe the improvement compared to Part 1 for all of the digits, as well as the overall FID score.

2.5 Recurrent Relational Network

The RRN architecture is the same as in part 1.

The RRN was trained as in part 1 and the best model was found at the end of the 97th epoch.

The performance metrics for the RRN are as follows:



(a) Losses **(b)** Sudoku Accuracies **(c)** Digit Accuracies

Best Model	Loss	Sudoku Acc	Digit Acc
Train	0.0812	0.9671	0.9962
Val	0.0858	0.9600	0.9939
Test	0.0873	0.9630	0.9941

Figure 5. Evaluation Metrics for RRN

We can see a clear improvement in Sudoku Accuracy from the 0.25% validation sudoku accuracy obtained from the best model before. Due to the improved clustering in part 3, our digit accuracy has gone up to 99.39% on the validation set in the best model, and due to the improved accuracy, we are able to get sudoku accuracy of upto 96% on the validation set in this model.

REFERENCES

1. https://arxiv.org/pdf/1707.02131.pdf
2. https://ece.uwaterloo.ca/ z70wang/research/ssim/
3. https://arxiv.org/abs/1411.1784
4. https://github.com/soumith/ganhacks
5. https://arxiv.org/abs/1711.08028
6. https://arxiv.org/pdf/1707.02131.pdf
7. https://arxiv.org/pdf/1807.06653.pdf