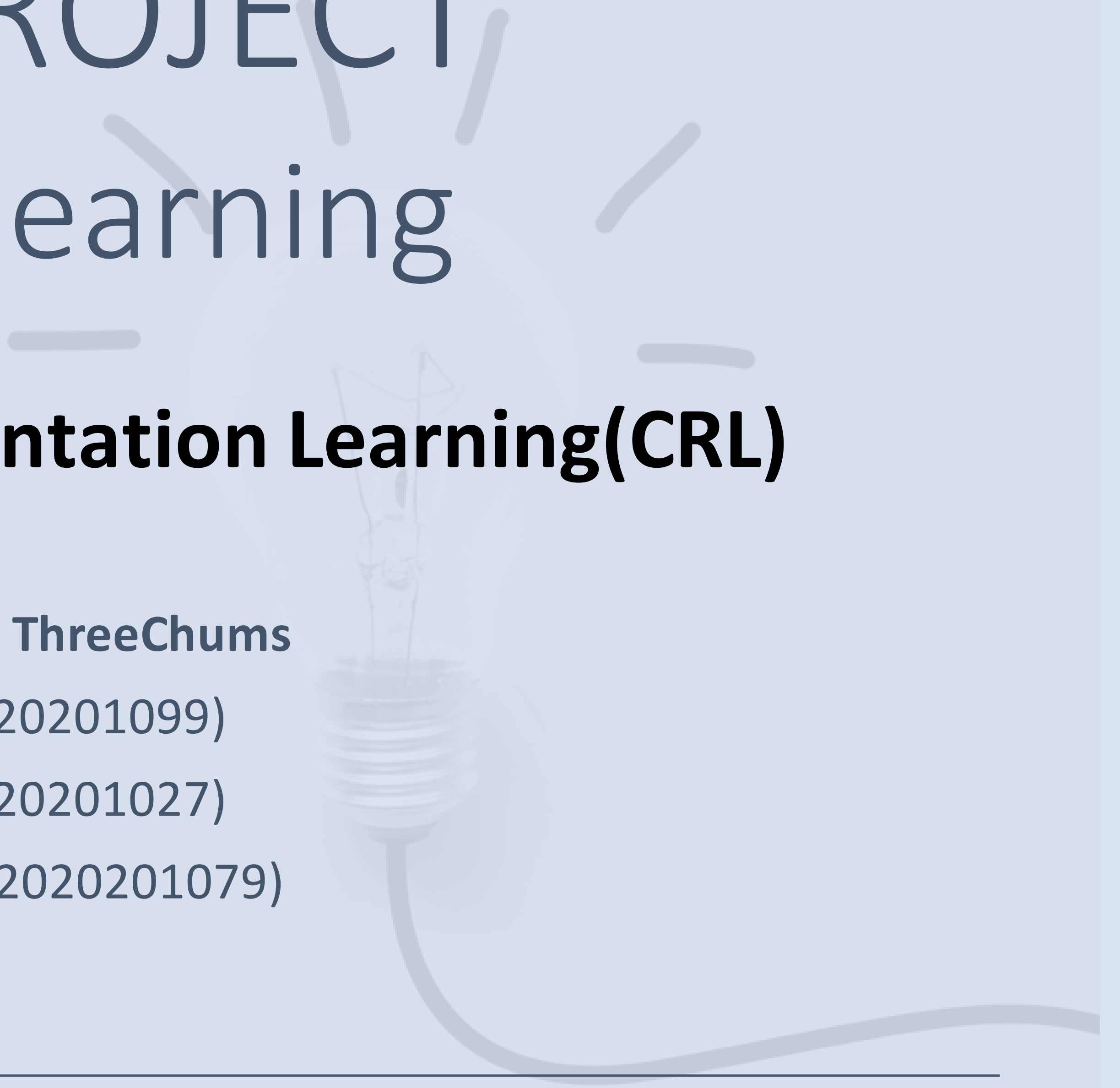
SMAI PROJECT on Deep learning



Common Representation Learning(CRL)

presented by Team - ThreeChums

- Anchal Soni (2020201099)
- MK Utkarsh (2020201027)
- Varun Nambigari (2020201079)

Common Representation Learning(CRL)

learning a common representation for multi-view data, wherein the different modalities are projected onto a common subspace

For example, task of abstract scene recognition in a movie

In this paper we construct a novel step-based correlation multi-modal CNN (CorrMCNN) which can reconstruct one view of the data given the other.

Two popular techniques used for CRL

- canonical based approaches(CCA)

  Drawbacks:

  - scalability issues
  - poor performance in reconstrction

- Autoencoder based methods

  A) Deep neural networks that try to optimize two objective functions

  B) First is a compressed hidden representation of data in a low dimensional vector space.

  C) Second is reconstruct the original data from the compressed low dimensional subspace.

Multi Modal Auto-encoders MAE

- These are two channeled AE that specifically performs two types of reconstruction.
  - First is self reconstruction of view from itself
  - Another is cross reconstruction where one view is reconstructed given the other.
- These reconstruction objectives provide MAE the ability to adapt towards transfer learning tasks as well.

CorrNet is an advanced MAE

- by introducing a correlation term in the objective function that tries to maximize the correlation between the hidden representation of different views.
- Limitations of corrNet :
  - usage of simple neural layers for encoding and decoding
  - using the final hidden representations in the correlation loss function.

Main contribution of the paper and Introduction of CorrMNCNN

- convolutional layers in encoding phase and deconvolution in the decoding stage.
- Batch normalization in the intermediate layers along with tied weights architecture.
- Instead of using final hidden representations in the correlation loss, we enforce correlation computation at each intermediate layer. We further experiment with reconstruction of hidden representation at every individual step.

For training the proposed model, we target the following Goals and losses in our objective function:

- Minimize the self-reconstruction error.
- Minimize the cross reconstruction error at each interme-diate step.
- Introduce batch-normalization at the intermediate dense neural layers.
- Maximize the correlation between the hidden representation of both views at each encoding step.

Architecture of CorrMNCC

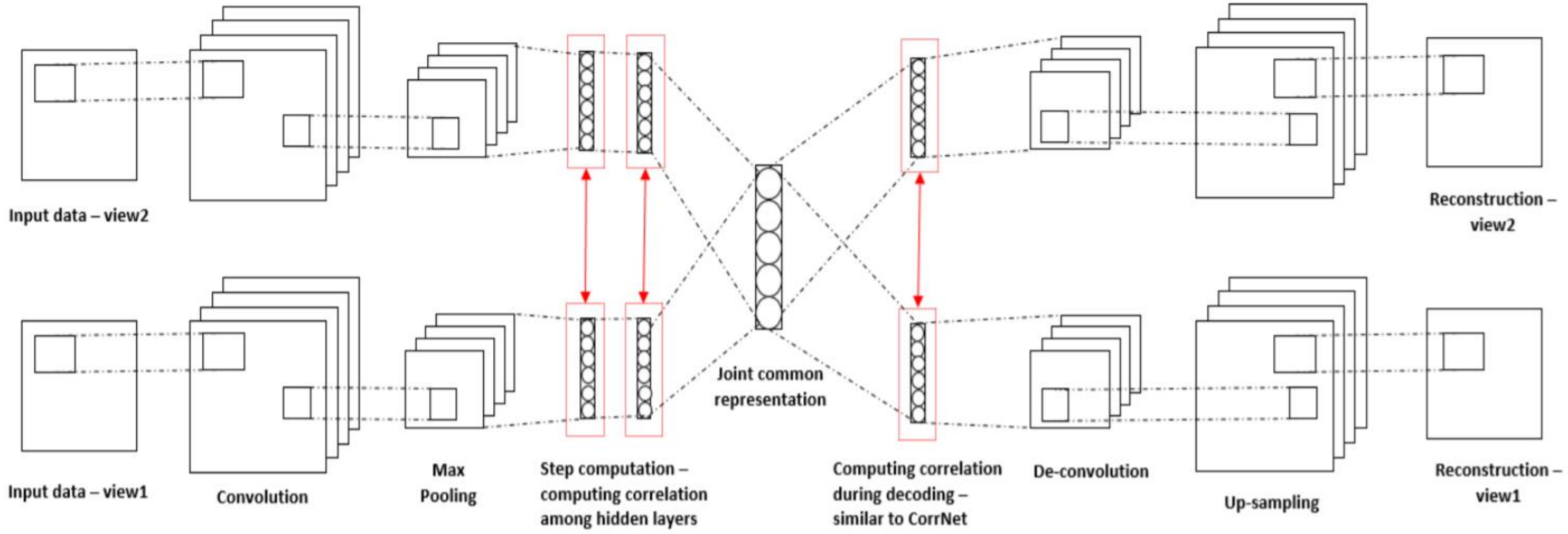

Input data – view2
Reconstruction – view2
Input data – view1
Convolution
Max Pooling
Step computation – computing correlation among hidden layers
Joint common representation
Computing correlation during decoding – similar to CorrNet
De-convolution
Up-sampling
Reconstruction – view1

Fig. 1. Overview of the CorrMCNN. The bidirectional arrows shows the step correlation computation and cross-reconstructions at the intermediate steps.

Explaination

- Input data is passed on to two channels. Each input having 28x14 dim image
- Further adding convolution and max-pooling to obtain a lower dimensional representation of the data containing imp features like edges, boundaries shapes etc.
- Then We add dense neural layers so that the interaction between the hidden representations can be maximized. We also added dropout to these intermediate dense layers.
- Finally, the output of the dense neural layers is used to obtain a joint common representation.
- To reconstruct the original input from the joint common representations, the projections are passed through a dense layer followed by deconvolution and up-sampling.
- The final outputs we get are the reconstructions of the input view 1 and input view 2.

Self and reconstruction Losses:

L is the mean square error function

$$L_1 = \sum_{i=0}^{N} L(z_i, g(h(z_i))$$

$$L_2 = \sum_{i=0}^{N} L(z_i, g(h(x_i))$$

$$L_3 = \sum_{i=0}^{N} L(z_i, g(h(y_i))$$

$$L_4 = \sum_{k=0}^{K} \sum_{i=0}^{N} L(h(x_i^k), h(y_i^k))$$

$$L_5 = \sum_{i=0}^{N} L(g(h(x_i)), g(h(y_i))$$

Correlation losses:

$$L_6 = \lambda \; corr(h(X), h(Y))$$

$$L_7 = \sum_{k=0}^{K} \lambda_k \; corr(h(X^k), h(Y^k))$$

Lamda is the regularization parameter

$$\mathrm{corr}(h(X), h(Y)) = \frac{\sum_{i=1}^{N}(h(\mathbf{x}_i) - \overline{h(X)})(h(\mathbf{y}_i) - \overline{h(Y)})}{\sqrt{\sum_{i=1}^{N}(h(\mathbf{x}_i) - \overline{h(X)})^2 \sum_{i=1}^{N}(h(\mathbf{y}_i) - \overline{h(Y)})^2}}$$

Notations used:

- Given the input as $zi = \{x_i; y_i\}$, where z_i is the concatenated representation of input views x_i and y_i
- g, h are non-linearities generally taken as sigmoid or reLU
- $g(h(x^k_i))$ and $g(h(y^k_i))$ are the hidden representations
- K represents the kth intermediate hidden layer
- the value of k is 2 during encoding
- 1 in decoding

Loss function Description

LOSS NUMBER	USED FOR
Loss 1	**Self Reconstruction**
Loss 2 and Loss 3	**Cross reconstruction**
Loss 4	**Step Computation. Make the hidden layers similar**
Loss 5	**The Joint common representation should be similar for both views**
Loss 6 , Loss7	**Hidden layers and join representation must be correlated**

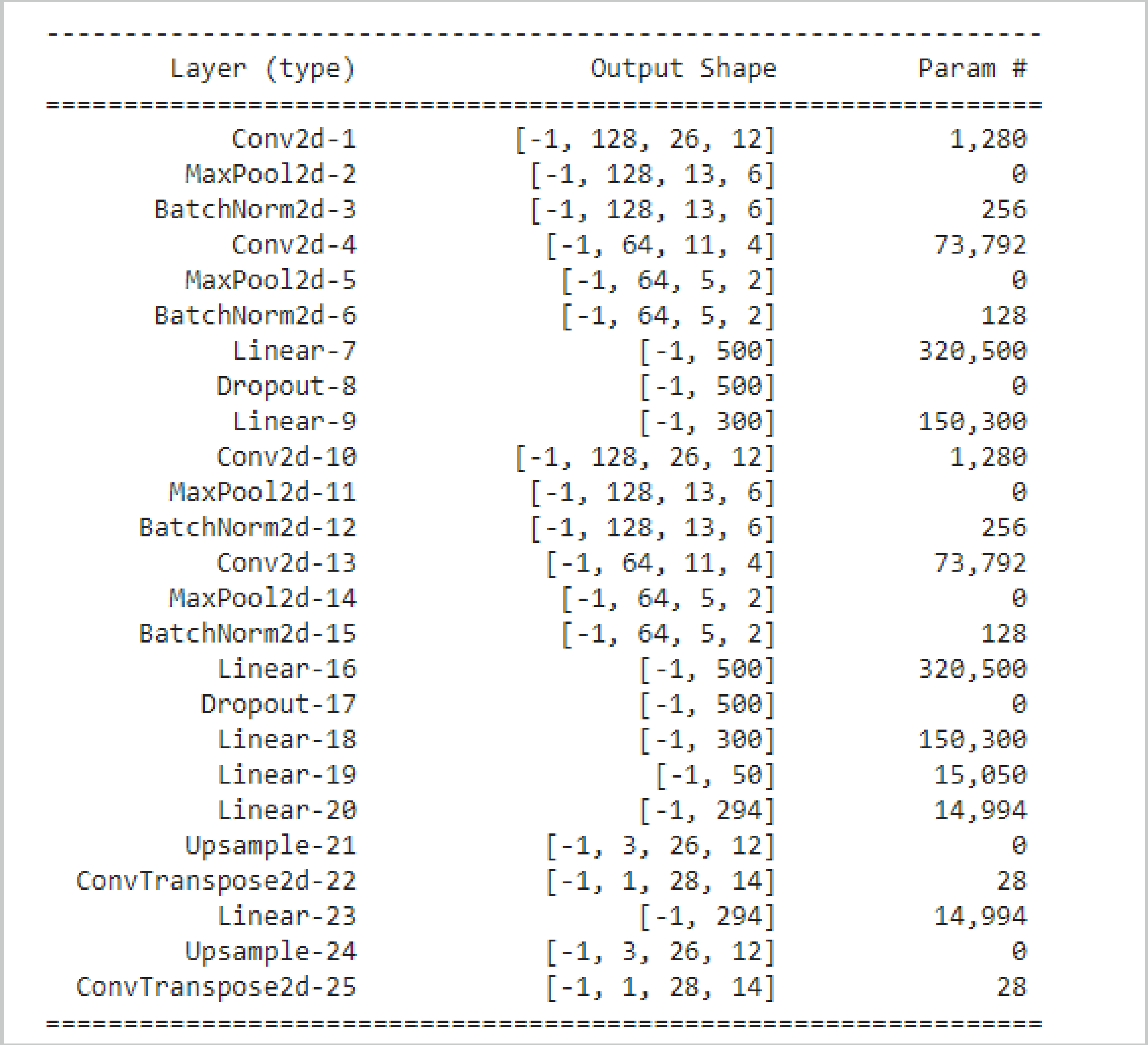
Finally, the CorrMCNN is optimized using the given objective function using adam as optimizer

$$L(\theta) = \sum_{i=0}^{5} L_i - \sum_{j=6}^{7} L_j$$

* where θ are the parameters of CorrMCNN
* Here, we minimize the self-reconstruction and cross-reconstruction
* and maximize the correlation between the views.

Model Summary

- Two input channels
- In each channel:
  * Two convolution layers
  * with MaxPooling layer
  * and Batch Norm layer
  * two fully connected layer
- Joint common representation with 50 dimension
- For each projection:
  * Upsampling
  * Deconvolution layer
- Two final reconstructed views



```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Conv2d-1          [-1, 128, 26, 12]           1,280
         MaxPool2d-2           [-1, 128, 13, 6]               0
       BatchNorm2d-3           [-1, 128, 13, 6]             256
            Conv2d-4            [-1, 64, 11, 4]          73,792
         MaxPool2d-5             [-1, 64, 5, 2]               0
       BatchNorm2d-6             [-1, 64, 5, 2]             128
            Linear-7                  [-1, 500]         320,500
           Dropout-8                  [-1, 500]               0
            Linear-9                  [-1, 300]         150,300
           Conv2d-10          [-1, 128, 26, 12]           1,280
        MaxPool2d-11           [-1, 128, 13, 6]               0
      BatchNorm2d-12           [-1, 128, 13, 6]             256
           Conv2d-13            [-1, 64, 11, 4]          73,792
        MaxPool2d-14             [-1, 64, 5, 2]               0
      BatchNorm2d-15             [-1, 64, 5, 2]             128
           Linear-16                  [-1, 500]         320,500
          Dropout-17                  [-1, 500]               0
           Linear-18                  [-1, 300]         150,300
           Linear-19                   [-1, 50]          15,050
           Linear-20                  [-1, 294]          14,994
         Upsample-21            [-1, 3, 26, 12]               0
  ConvTranspose2d-22            [-1, 1, 28, 14]              28
           Linear-23                  [-1, 294]          14,994
         Upsample-24            [-1, 3, 26, 12]               0
  ConvTranspose2d-25            [-1, 1, 28, 14]              28
================================================================
```

Parameters used

Dropout = 0.20
Batch size = 64

Value of lambda for correlation loss:

- lambda 1 = 0.02
- lambda2 = 0.003
- lambda3 = 0.05

No. of epochs

- 50

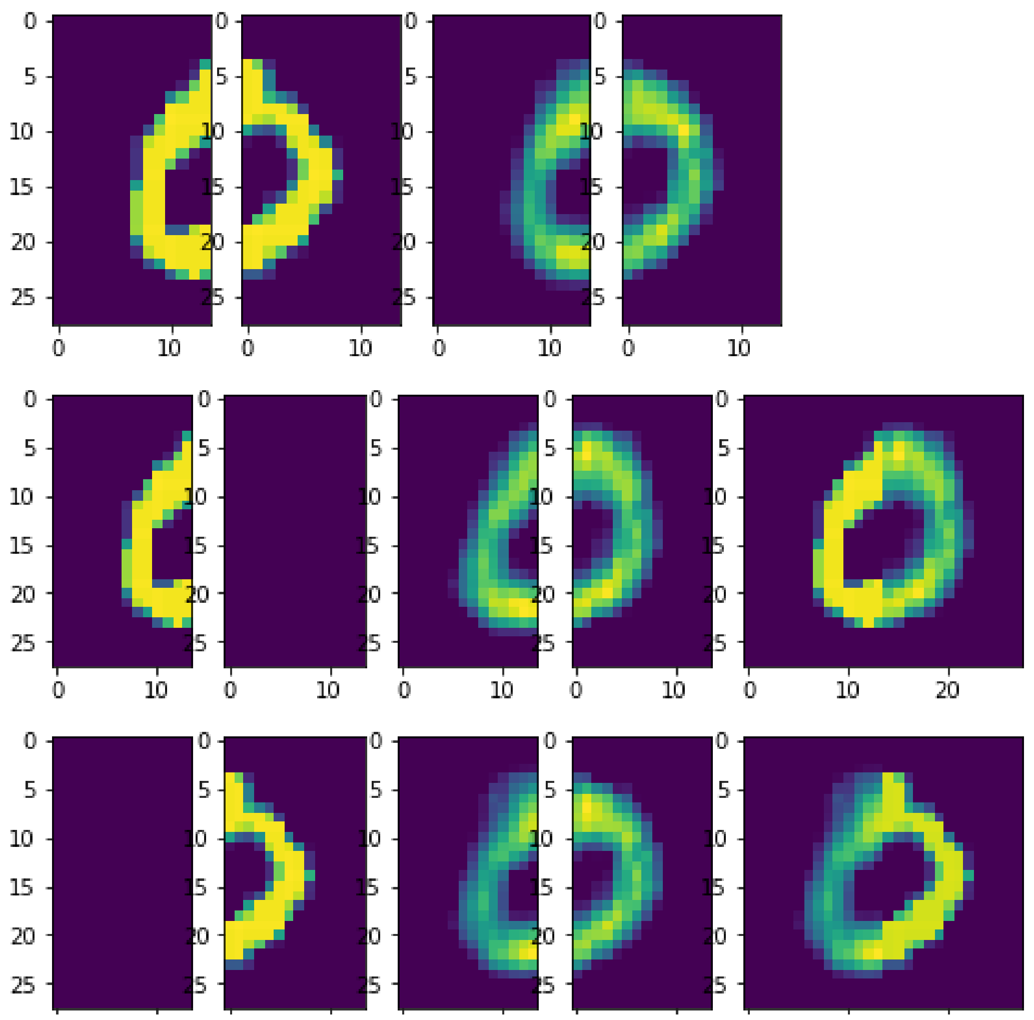
Optimizer used

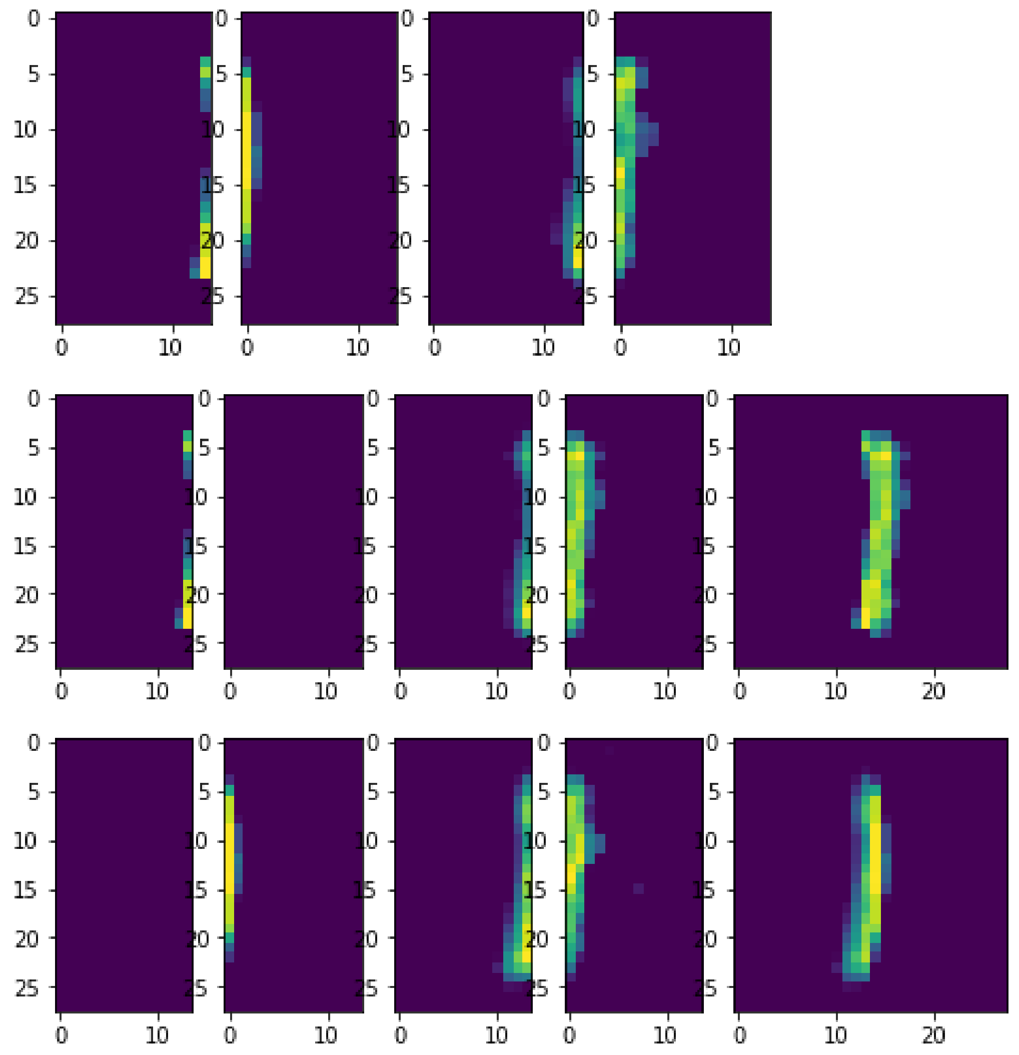
- Adam
- Learning rate = 0.01

Activation function used

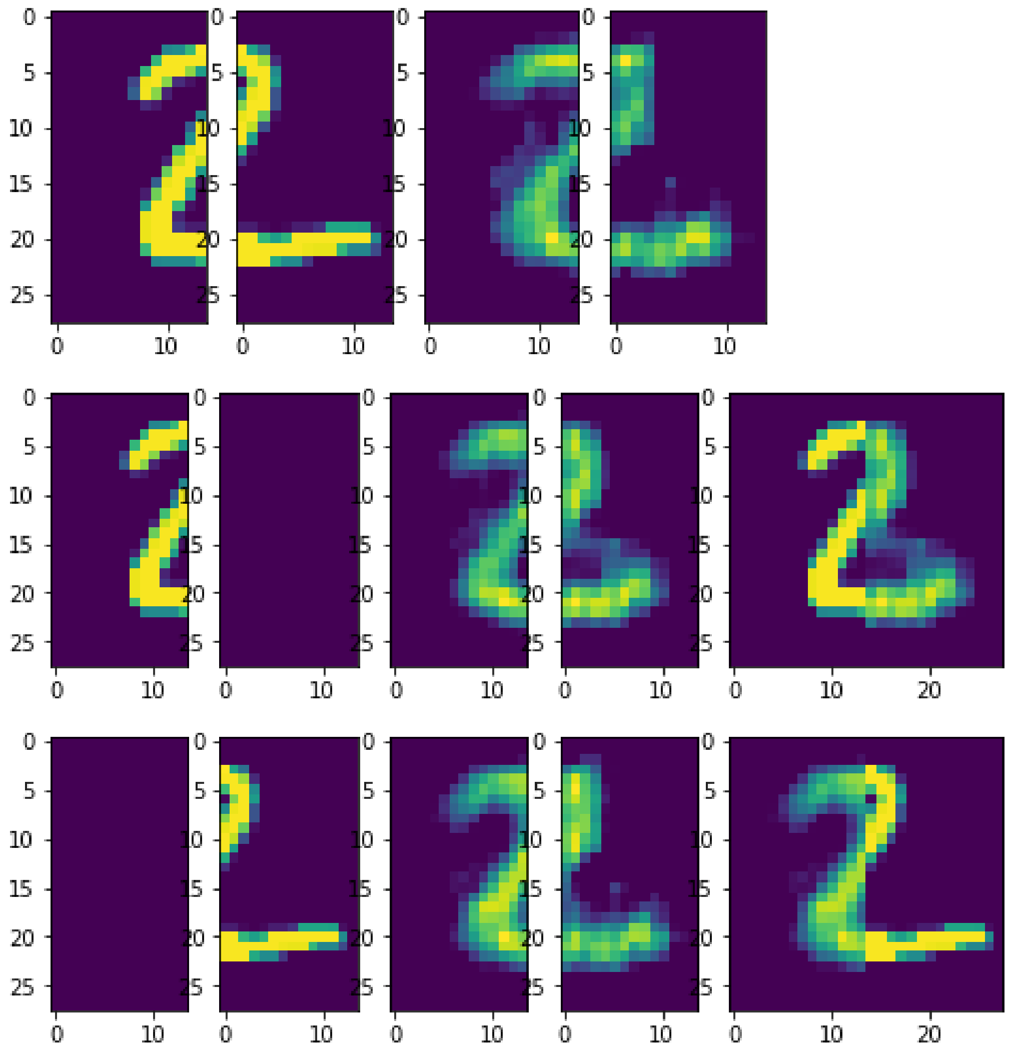
- ReLU

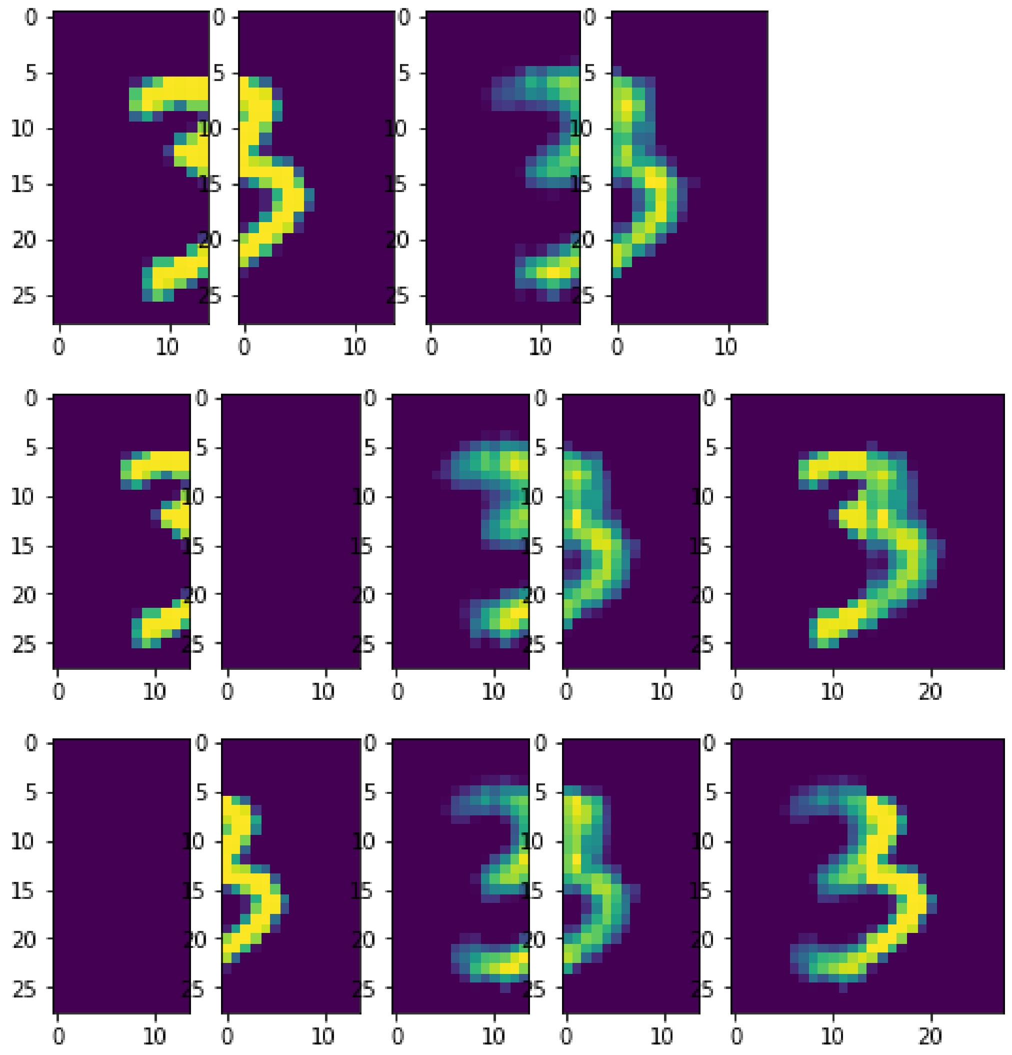
Reconstruction of '0' and '1'



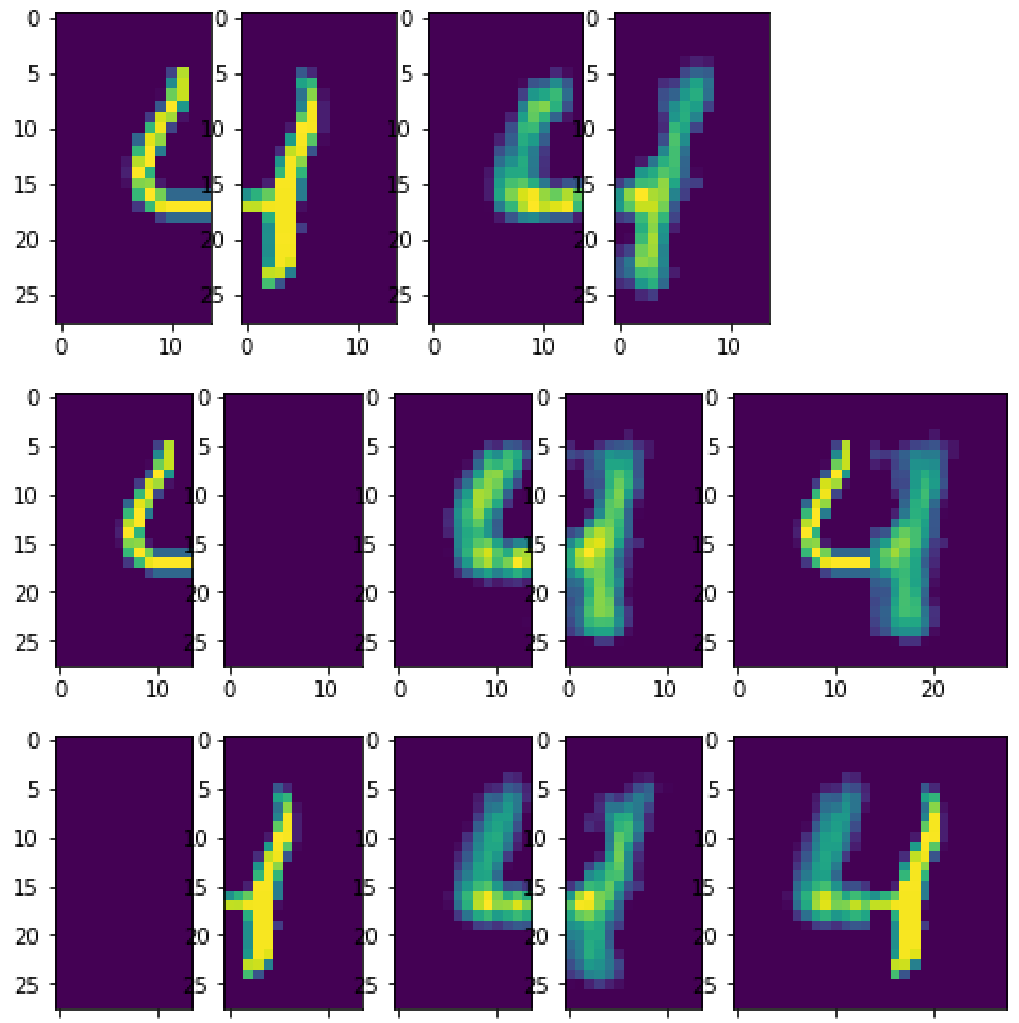


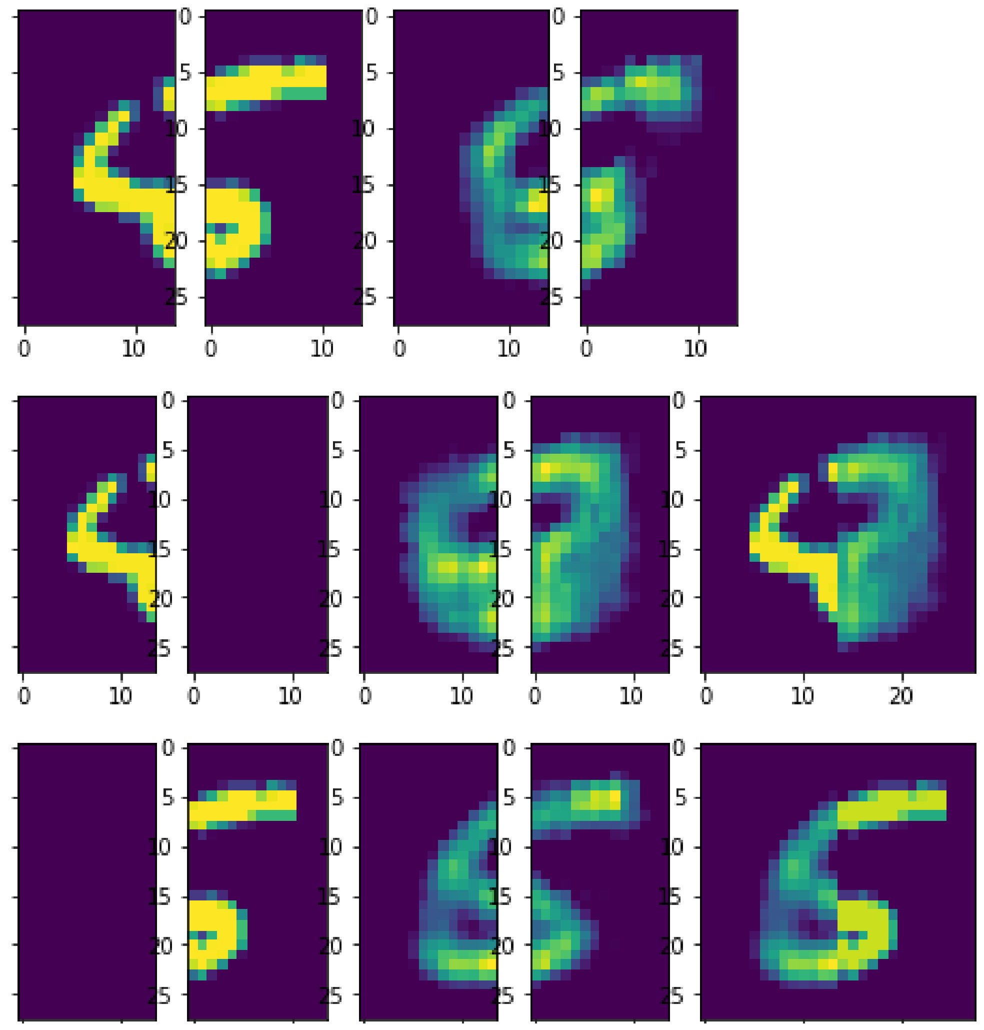
Reconstruction of '2' and '3'



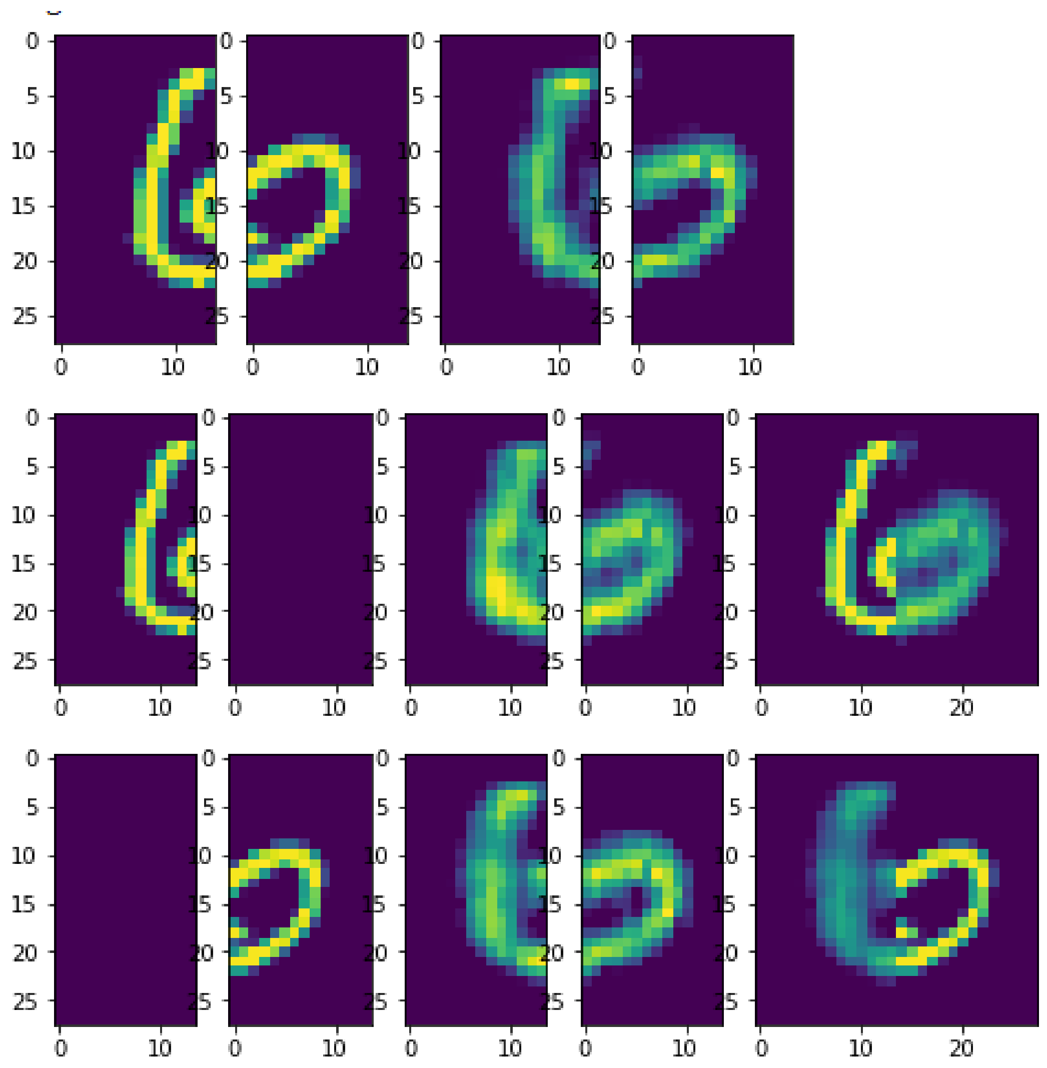


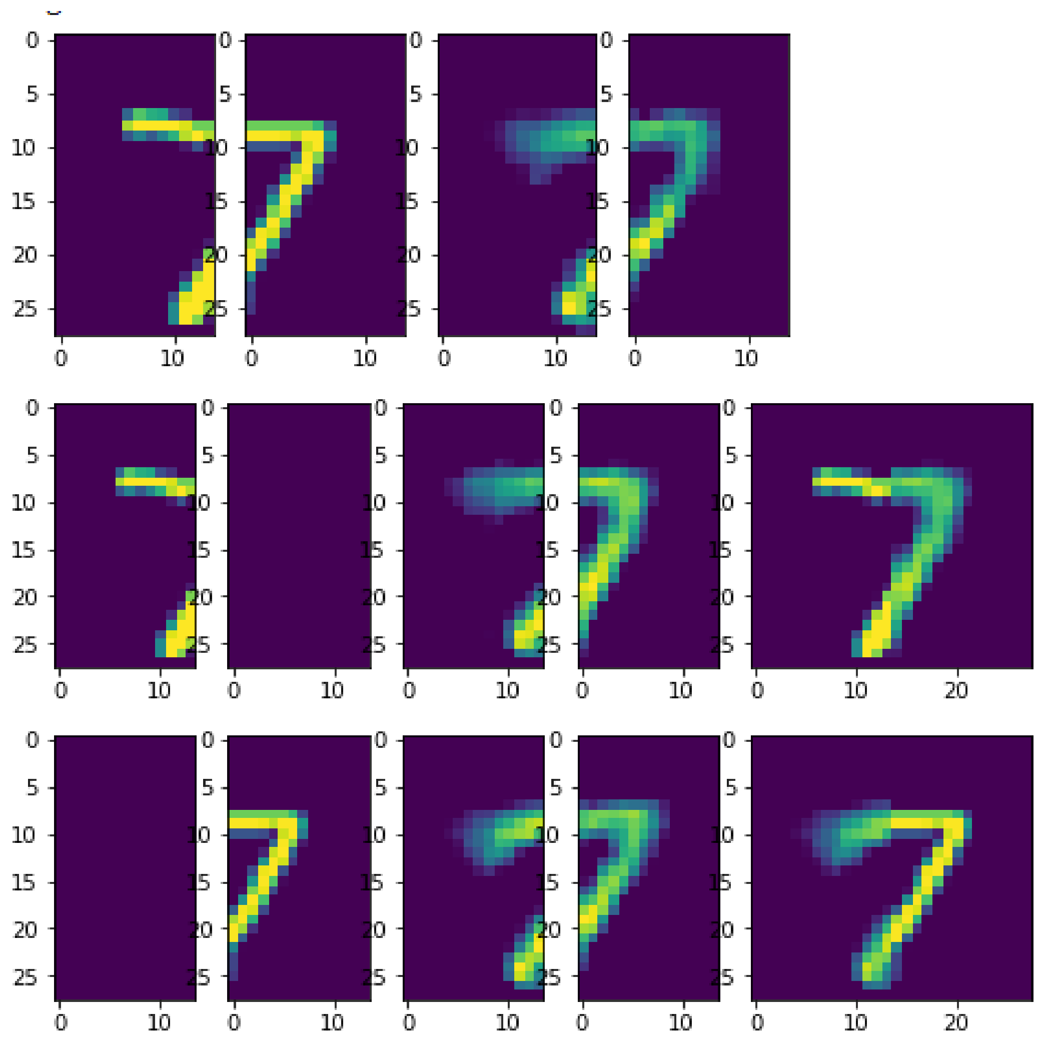
Reconstruction of '4' and '5'



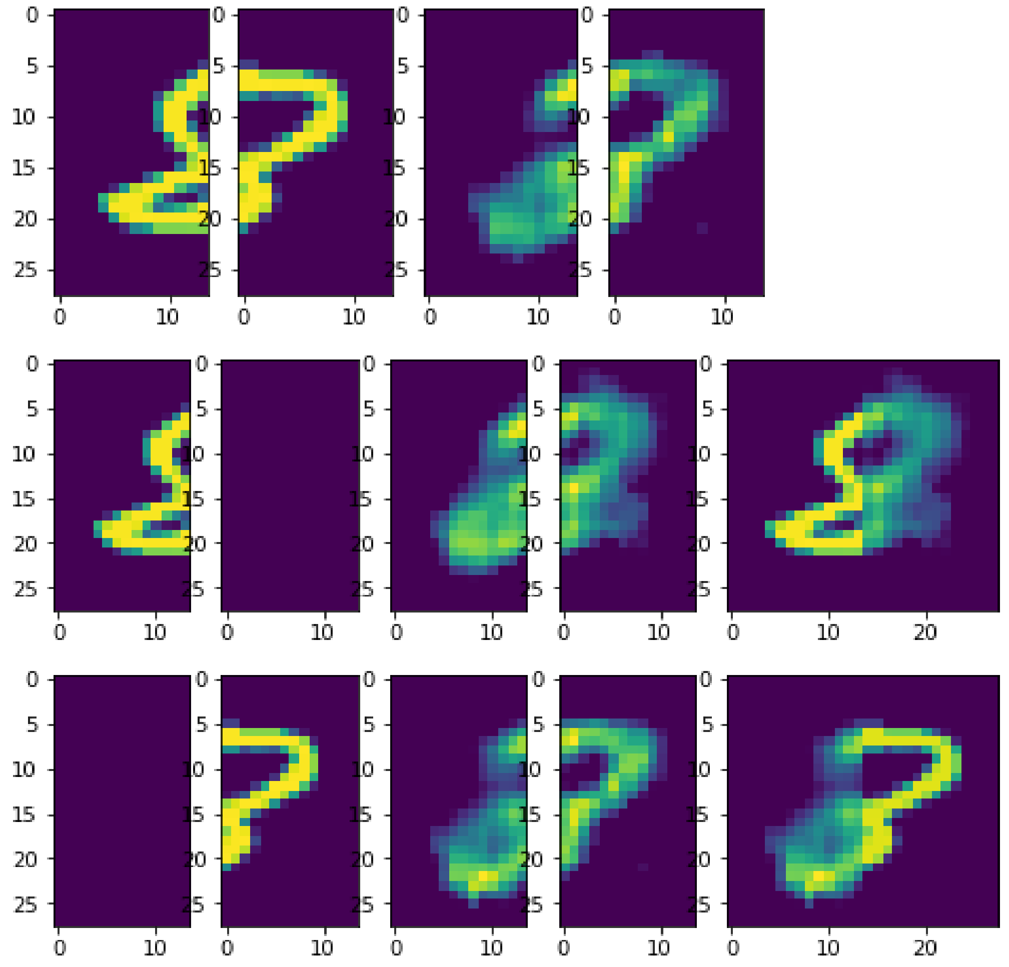


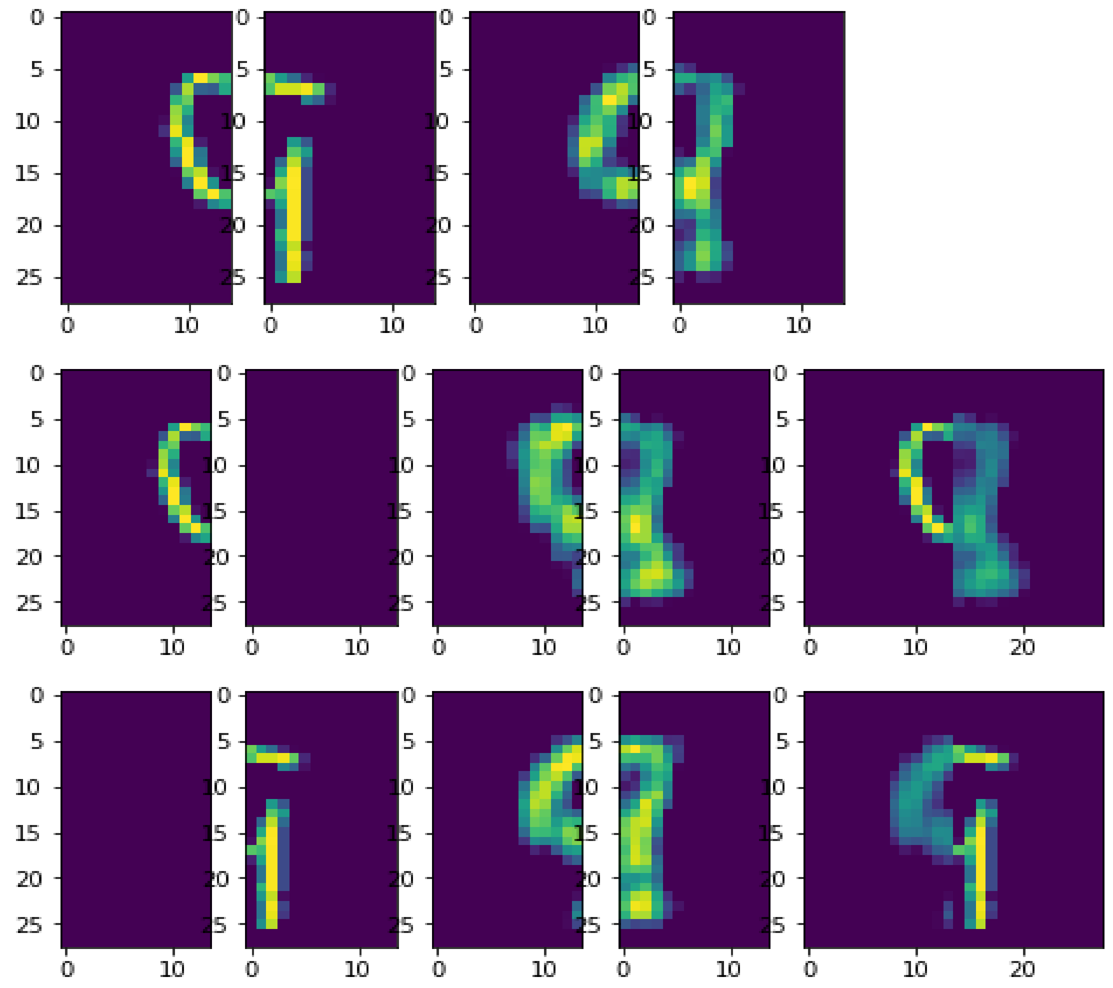
Reconstruction of '6' and '7'





Reconstruction of '8' and '9'





Problem of vanishing gradient

As more layers using certain activation functions are added to neural networks, the gradients of the loss function approaches zero, making the network hard to train.

Traditionally, high learning rates in deep neural networks resulted in vanishing gradient.

The result is that models with many layers will prematurely converge to a poor solution.

Solution - Batch Normalization and Use of reLU as activation function which is mathematically efficient instead of sigmoid.

Problem of overfitting

Overfitting occurs when you achieve a good fit of your model on the training data, while it does not generalize well on new, unseen data.

Regularization methods to reduce over-fitting.

Dropout technique is a very commonly used stochastic regularization technique which is also being used to prevent overfitting

Batch Normalization

Batch normalization has been used to increase the training rate providing us with better correlation values as compared to the previous papers.

It is a stabilizing mechanism for training a neural network by scaling the output of hidden layers to zero norm and unit variance.

This process of scaling reduces the change of distribution between neurons throughout the network and helps to speed up the training process.

Testing the model

- Trained a (2 layer fully connected MLP) classifier with input as hidden representation of the constructed using both the views.

- For testing constructed the hidden layer from only one view. Then classified based on the constructed hidden layer.

- Results:

  Classification accuracy from only left view: 74.25%
  Classification accuracy from only right view: 77.68%

Individual Contribution:

- Anchal Soni (2020201099) - Data preprocessing. Built an Autoencoder (First three losses)
- Utkarsh MK (2020201027) - Corr loss function, loss 4 to loss 7. Completed the basic model class.
- Varun Nambigari (2020201079) - Trained the model, plotted the results. Tested using a classifier.

THANK YOU

Jupyter notebook link:
https://colab.research.google.com/drive/1hUFOlHfxVyqbVWP0Spciley-K-sJiLI-?usp=sharing

Research paper link:
https://arxiv.org/pdf/1711.00003.pdf

Buzz words:

- Vanishing gradient-situation where NN is unable to propagate useful gradient information from the output end of the model back to the layers near the input end of the model.The result is the general inability of models with many layers to learn on a given dataset, or for models with many layers to prematurely converge to a poor solution.
- Exploding gradient- Exploding gradients are a problem where large error gradients accumulate and result in very large updates to neural network model weights during training.This has the effect of your model being unstable and unable to learn from your training data.
- Batch normalization- like normalization scales input into common scale similarly batch norm scales the neurons weight into a common scale so that no particular neuron weight dominates the network output accelerates training, in some cases by halving the epochs or better, and provides some regularization, reducing generalization error.
  is a technique to standardize the inputs to a network, applied to ether the activations of a prior layer or inputs directly.
- Optimisers: Optimizers are algorithms or methods used to change the attributes of the neural network such as **weights** and **learning rate** to reduce the losses. Optimizers are used to solve optimization problems by minimizing the function.