Dynamic Routing Between Capsules

-Geoffrey E. Hinton

2018A1PS0006P, Hemant Bhartiya
2018A2PS0147P, Harshita Gupta

Project Repository:
https://github.com/HB5101/CapsNet_Project.git

Paper Summary

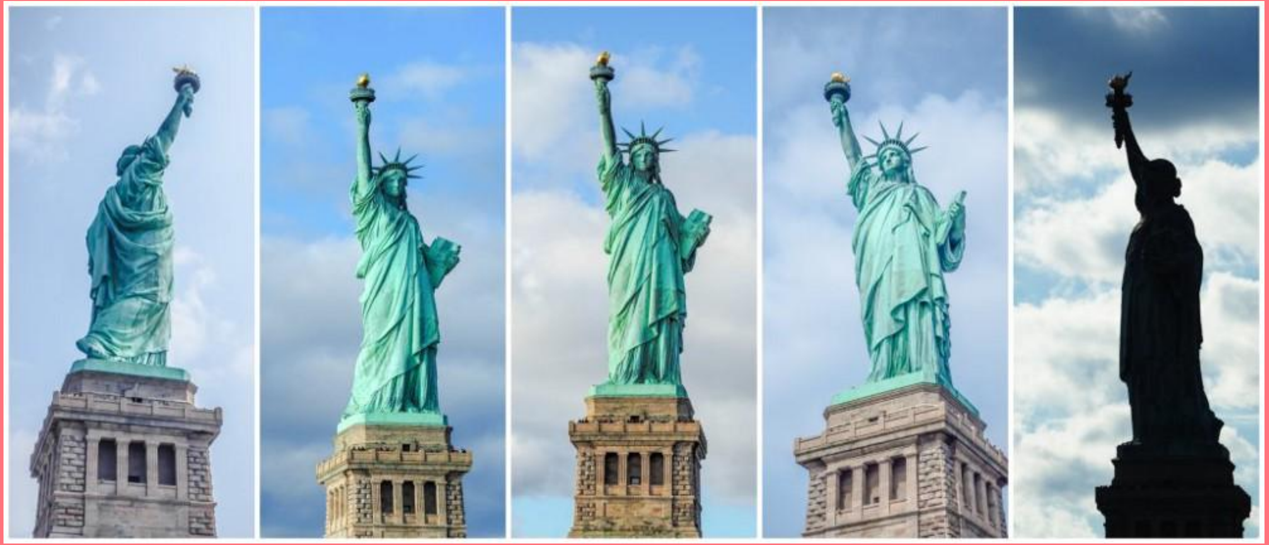
- The paper introduces a new concept of Convolutional Neural Networks known as Capsule Networks
- A capsule is a group of neurons whose activity vector represents the instantiation parameters of a specific type of entity such as an object or an object part.
- We use the length of the activity vector to represent the probability that the entity exists and its orientation to represent the instantiation parameters
- Capsules encapsulate all important information about the state of the feature they are detecting in vector form.

Intuition behind CapsNet

- Computer graphics deals with constructing a visual image from some internal hierarchical representation of geometric data.
- The internal representation is stored in computer’s memory as arrays of geometrical objects and matrices that represent relative positions and orientation of these objects.
- Then, special software takes that representation and converts it into an image on screen known as rendering.
- Inspired by this idea, Hinton argues that brains, in fact, do the opposite of rendering. He calls it inverse graphics: from visual information received by eyes, they deconstruct a hierarchical representation of the world around us and try to match it with already learned patterns and relationships stored in the brain.

The Key Concept: Representation of objects in the brain does not depend on view angle.

Consider the image below:



Here, you can easily recognize that this is the Statue of Liberty, even though all the images show it from different angles. This is because internal representation of the Statue of Liberty in your brain does not depend on the view angle.

Drawback of CNNs

The paper claims that CapsNet perform better than the existing CNNs. These are basically alternatives to them.

The main drawback of CNN as quoted in the paper can be understood from the following example-

We have the face oval, two eyes, a nose and a mouth. For a CNN, a mere presence of these objects can be a very strong indicator to consider that there is a face in the image. Orientational and relative spatial relationships between these components are not very important to a CNN.

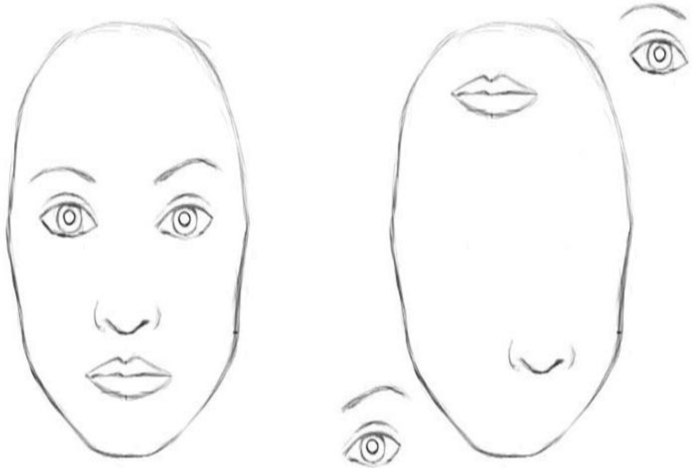
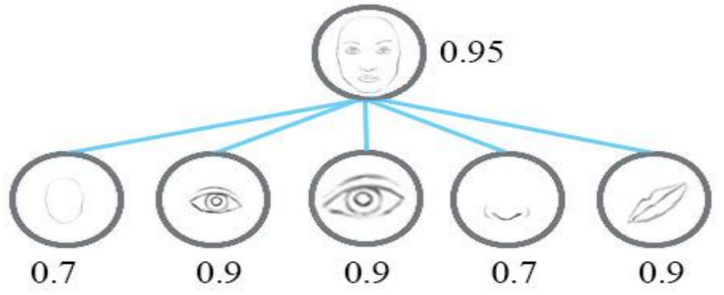


fig: Both the image is similar to CNNs.

CNN vs CapsNet

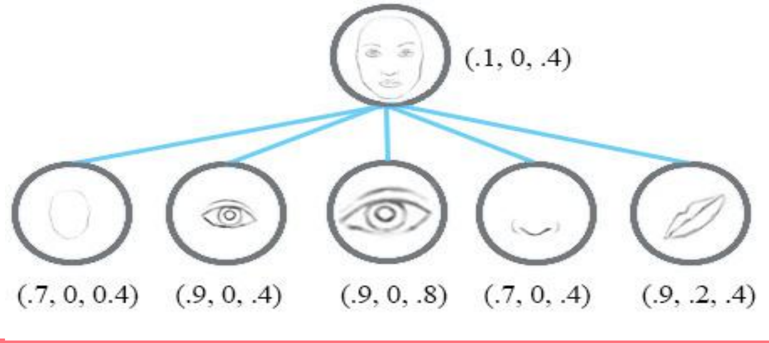
Feature extraction of a face by CNN:

fig: Neurons outputs a scalar containing likelihood in CNN.



Feature extraction of a face by CapsNet:

fig: Capsule outputs a vector containing likelihood, orientation, size in CapsNet.



In Capsule Network, we have equivariance which is the detection of objects that can transform to each other. It may be stated as, CNN model uses multiple neurons and layers in capturing different variation in features , whereas for the same purpose capsule network share the same capsule.
This property helps Capsule Network to extrapolate possible variants more effectively with less training data.

CNN vs CapsNet

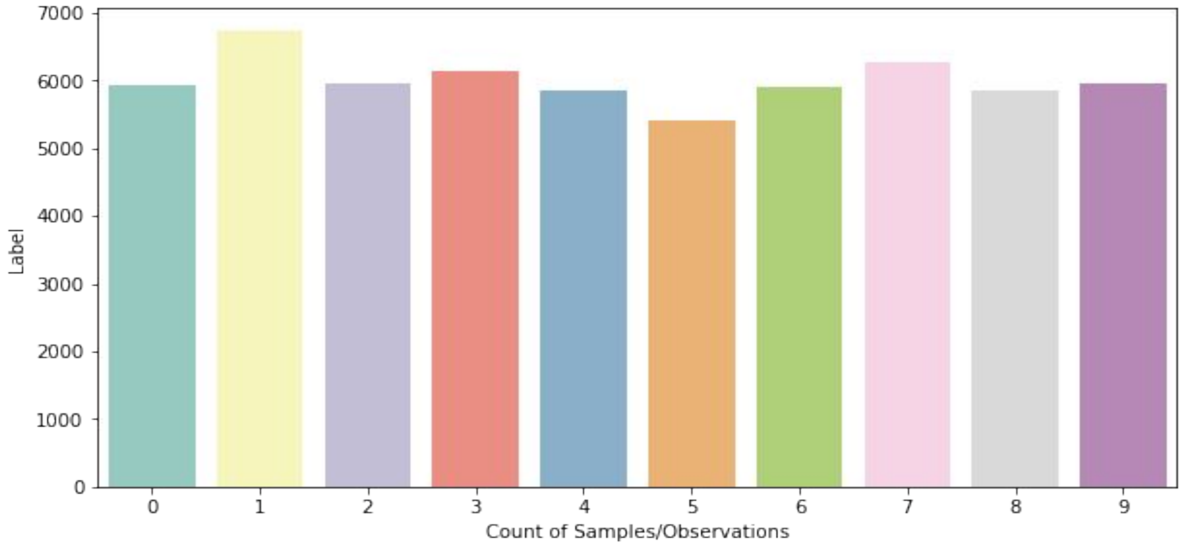
Also, we use dynamic routing to compute the final output of a capsule instead of only using backpropagation with cost function.We compute Cij to quantify the connection between a capsule and its parent capsules.Conceptually, Cij measures how likely capsule i may activate capsule j.

In a CNN, max pooling handles translational variance, but this approach keeps only the most dominating feature and throws away the others. Capsules maintain a weighted sum of features from the previous layer. Hence, it is more suitable in detecting overlapping features.

For all the experiments, we used the following as per the paper-

Data set: affNIST dataset, based on MNIST dataset

The dataset has 60K and 10K images for training and testing respectively. Dataset has 10 classes.



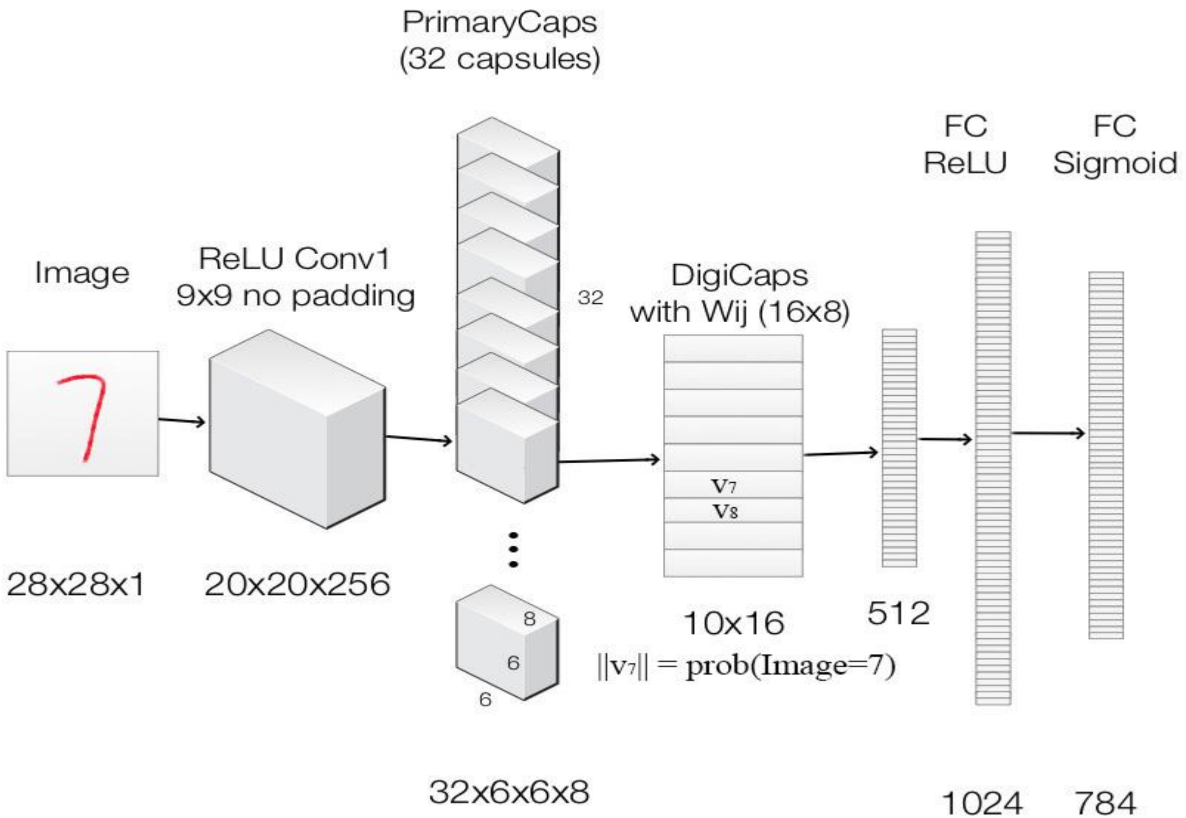
DATASET

fig: EDA of training dataset

Data Processing/ Hyper-parameters

Data Augmentation: As per the paper, it suggested to use 28 × 28 images that have been shifted by up to 2 pixels in each direction with zero padding. No other data augmentation/deformation is used.

Hyperparameters: As per the paper, it suggested to use "Adam" optimizer, with its TensorFlow default parameters, including the exponentially decaying learning rate, to minimize the sum of the margin losses.



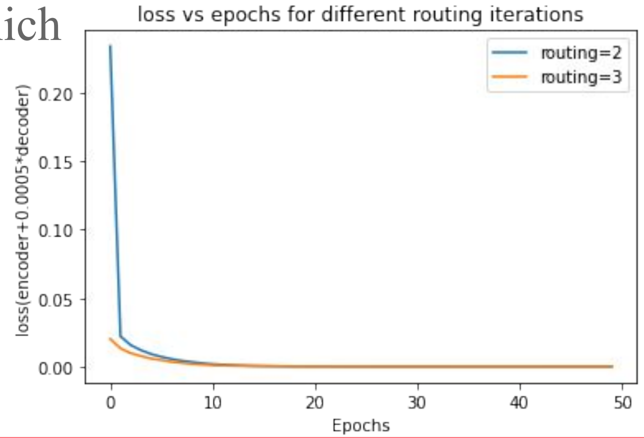
MODEL

fig: CapsNet Architecture

Routing iterations

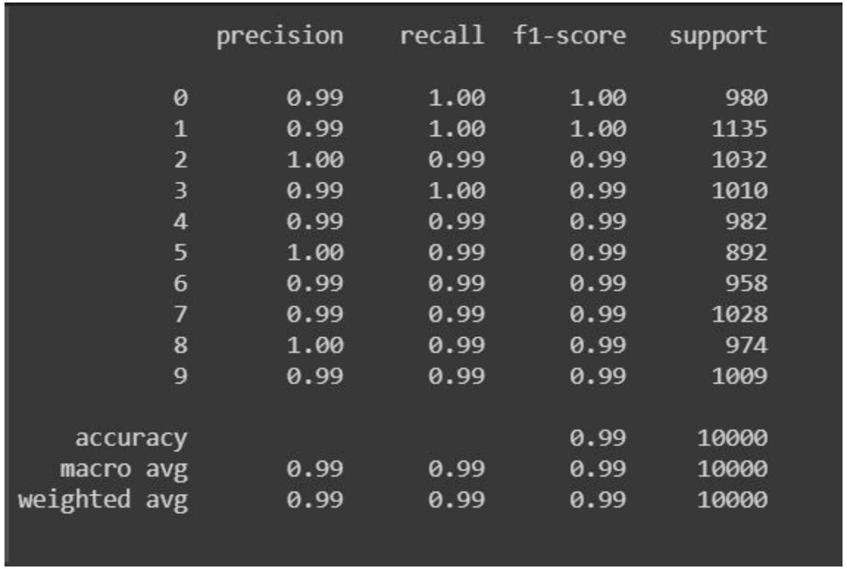
In Capsule Network, we use iterative dynamic routing to compute the capsule output by calculating an intermediate value Cij (coupling coefficient). Depending on the number of iterations, different Cij matrix is obtained and hence capsules of output layer are activated accordingly.

While, training with routing iterations 2 and 3, we observed that with 2 routings, loss was initially high which drastically dropped just after few initial epochs, whereas when number of routings was increased to 3, model started with low loss which eventually decreased slowly.

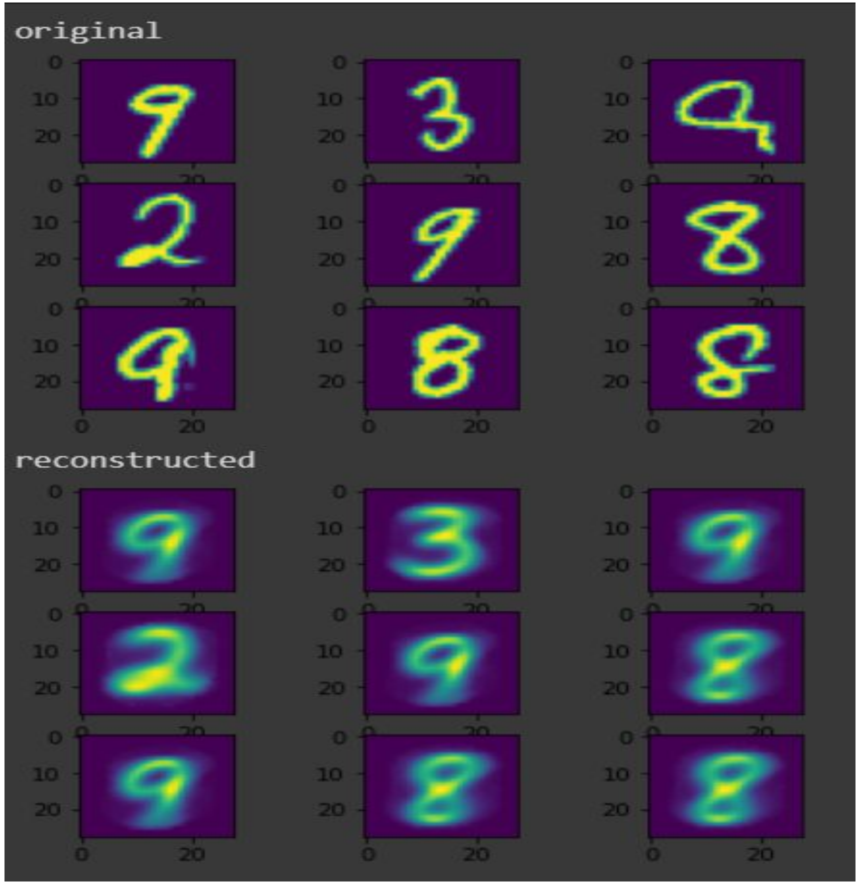


Qualitative Results

Routing Iteration=2

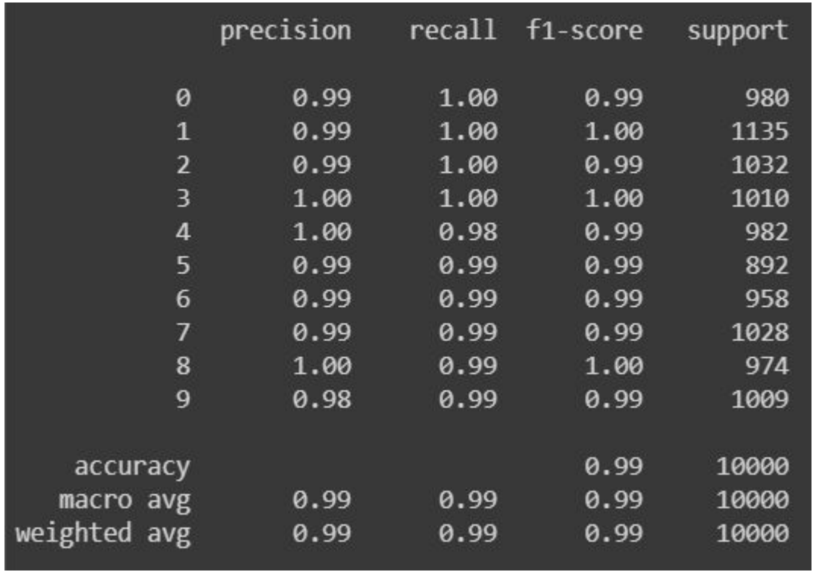


```
              precision    recall  f1-score   support

           0       0.99      1.00      1.00       980
           1       0.99      1.00      1.00      1135
           2       1.00      0.99      0.99      1032
           3       0.99      1.00      0.99      1010
           4       0.99      0.99      0.99       982
           5       1.00      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       1.00      0.99      0.99       974
           9       0.99      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```

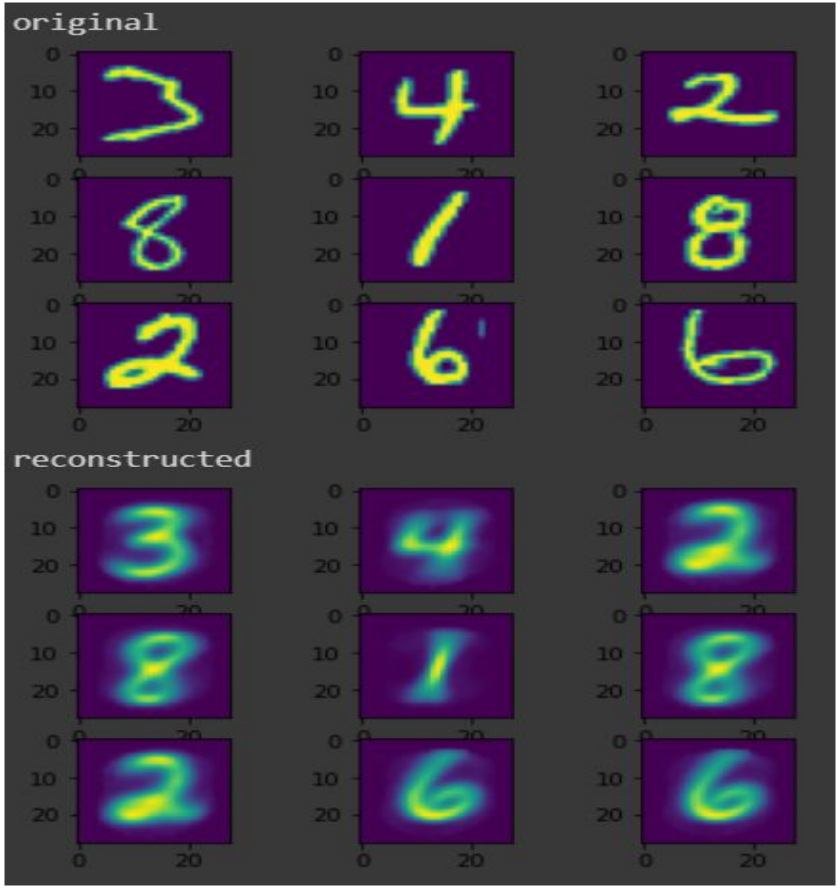


Qualitative Results

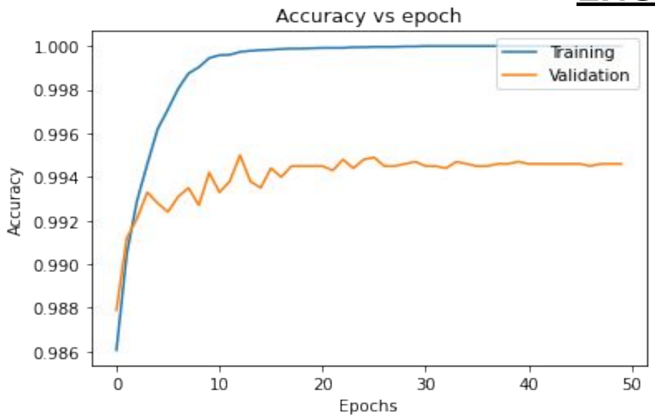
Routing Iteration=3

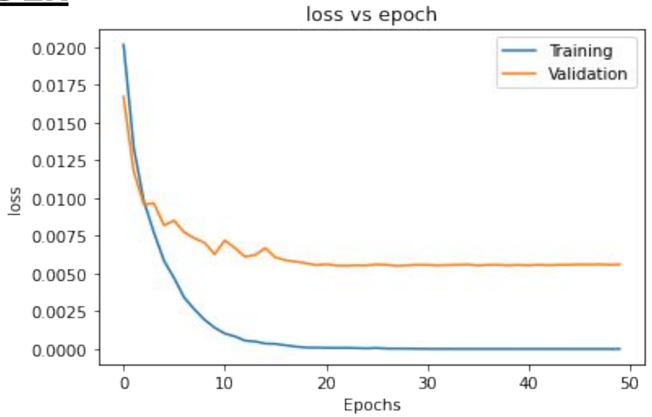


```
              precision    recall  f1-score   support

           0       0.99      1.00      0.99       980
           1       0.99      1.00      1.00      1135
           2       0.99      1.00      0.99      1032
           3       1.00      1.00      1.00      1010
           4       1.00      0.98      0.99       982
           5       0.99      0.99      0.99       892
           6       0.99      0.99      0.99       958
           7       0.99      0.99      0.99      1028
           8       1.00      0.99      1.00       974
           9       0.98      0.99      0.99      1009

    accuracy                           0.99     10000
   macro avg       0.99      0.99      0.99     10000
weighted avg       0.99      0.99      0.99     10000
```



ENCODER

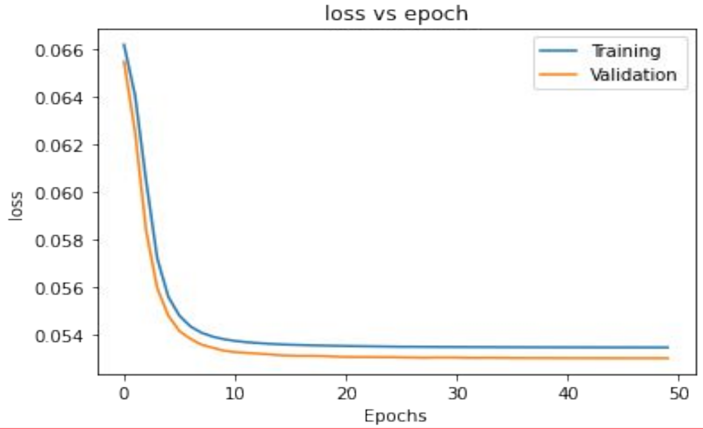




Quantitative Results

Routing iteration=3

DECODER



Observations

- With different number of routings we have different losses for initial epochs, but after a sufficient number of epochs losses converges to the same range.
- As seen from the loss graph for different iteration, routing=3 fits better and thus gives better prediction per class.

Routing 3	Training Loss	Validation Loss	Accuracy/ Loss
Encoder	0.000002	0.0056	99%
Decoder	0.054	0.054	0.054

- For routing=2, training speed observed was ~285s/epoch and for routing=3 it was ~448s/epoch
- As given, the learning rate decays exponentially.

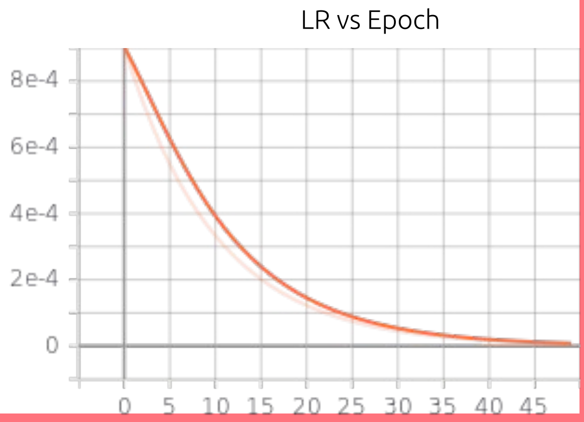


fig: LR decays exponentially.

Remarks

- Our reconstruction image doesn’t match up with that of the paper’s as we trained only for 50 epochs.
- According to paper, the CapsNet model has 8.2M parameters with reconstruction subnetwork and so as ours.
- In general more routing iterations increases the network capacity and tends to overfit to the training dataset
- The paper itself is not very detailed and hence it leaves some open questions about specifics of the network implementation that are as of today still unanswered because the authors did not provide their code.

Things we didn’t do

- Language Modelling: The paper requires training for different routing iterations such as 2,3,5,7,10 for every 1,250 epochs. Training such a huge model for 4x1,250=5,000 epochs cannot be run on Colab and simply we did not have the computer to do so otherwise.

- Training on CapsNet : Running the network multiple times was not feasible on colab and would have been too costly on GCP/AWS instances.

- We didn’t do segmentation on highly overlapped digits as it required to generate a large dataset that would have taken days for continuous training.

“The pooling operation used in convolutional neural networks is a big mistake and the fact that it works so well is a disaster”

-Geoffrey E. Hinton

THANK YOU!