**Model:**

The dataset available with us has various features such as MR Delay, Gender, Dominant hand, Age, Education, SES, MMSE, CDR, ASF, eTIV and nWBV. For best prediction of dementia using the available data, we are required to do hierarchical analysis of all the data points available.

CNN (CovNets) and DNN often become inadequate when hierarchical data analysis is involved, therefore to hand this task, we use a more refined form of CovNets called CapNet or Capsule Networks. CapNets were devised to handle hierarchical modelling problems and is the right fit for our problem.

We must note that, CapNets in no way resembles the “Pooling” layers used in CovNets. Pooling in CovNets is used to reduce the details and thus increasing the speed of algorithmic runtime. However in CapNets never decrease the details, in fact, it takes even the small of details into account.

We must appreciate that Capsule Networks do not transfer individual activations of neurons from one layer to the next layer, but each capsule represents a small nested neural network that generates an entire vector instead. The total length of the output vector of a capsule encodes the likelihood that a certain feature has been found. The position of the length of the vector helps to represent the state of the feature detected (e.g. location, pose, scale).

The in fact, creates too many parameters and thus making the system costly, in terms of computational firepower.

We are aware of the fact that computation is not a problem due to the advent of GPUs and cloud computing, however, we intend to make this model as simple as possible to increase its reach in view of the humanitarian need of this.

In view of this, we propose the use of Modified Capsule Network Architecture (<https://link.springer.com/chapter/10.1007/978-3-030-34120-6_21>) in this scenario.

The additional steps in the use of this method is the use of matrix decomposition at the end of each capsule to reduce the parametric load and also the introduction of a shallow feature extraction with it.

**Algorithm (Modified Low Cost CapNets):**

We have a parent vector which is translated and rotated by a decomposed transformer matrix into a unit vector defined as .

Say, there are capsules, the sum of predictions from capsules are fed into the capsules, with a coupling coefficient .

The coupling coefficient is the most important factor in this algorithm. The pseudocode of CapNets is given below,

1: Ingest Data (*D*)

2: Calculate the parent vector, (Contains the encoded values of features).

3: Define two sub-capsules, *i* meaning Dementia is present and *j* meaning Dementia is absent.

4. **Run basic CovNet with ReLU activation** function to generate a vector, say *a.*

5: **Squash the vector** *a* into a vector to fit our feature set, squashing is done using the following formula:

6: Each squashed vector is a unit vector and is used in a capsule along with a **transformation matrix**  to explain the hierarchical relationship between the classes mentioned in the respective capsule.

7: Matrix decomposition of to get

8: is computed only once.

9: From each capsule, the results are forwarded to the next capsule along with a coupling coefficient .

10: This coupling coefficient is calculated using the activation function of the CapNet. Therefore, .

11: Now, CapNet gives great results due to the **dynamic routing** of the results across the layers. The algorithm for dynamic routing is given below:

A: Define ***a*** (Unit Vector)

B: Define ***r*** (Iterations, Scalar)

C: Define ***l*** (No. of layers, Scalar)

D: foreach capsule ***n*** in layer ***l*** , capsule ***m*** in layer ***(l+1)*** do ***memory*** <- 0

E: for ***r*** do:

E1: foreach capsule ***n*** in ***l*** do ***ki <- softmax (b,n)***

E2: for each ***ki***, do ***ki’ <- KPCA (ki)***

E3: foreach capsule ***m*** in ***(l+1)*** do ***sj <-* ( is calculated using the decomposed )**

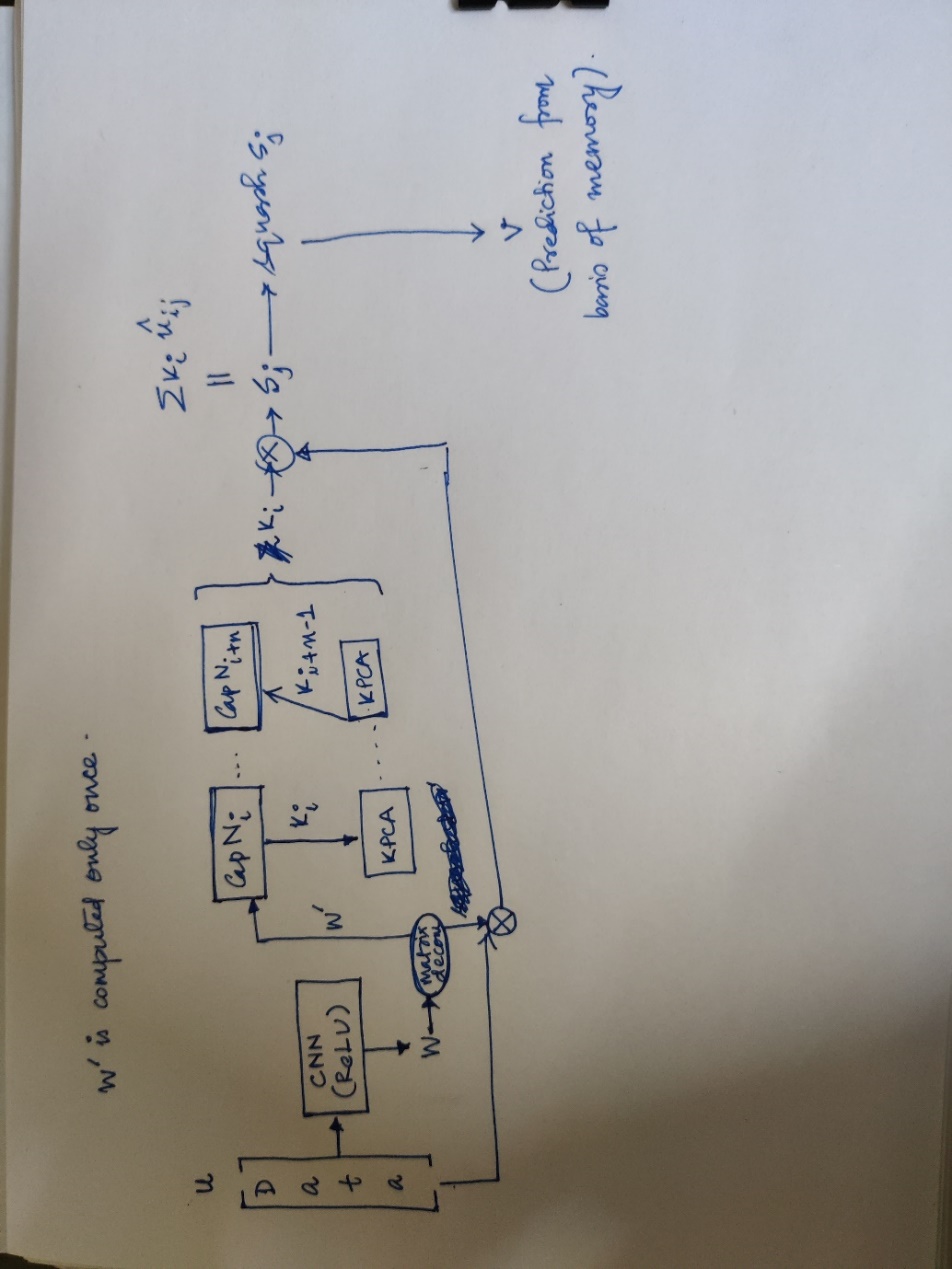
E4: for each ***sj*** do ***sj’ <- KPCA (sj)***

E3: foreach capsule ***m*** in ***(l+1)*** do ***a <- squash (sj’)***

E4: foreach capsule ***n*** in ***l***, ***m*** in ***(l+1)*** do ***memoryij <- memoryij + . v***

F: Return ***v***.

**Diagram:**



**Accuracy and comparison:**

