

A REPORT
ON
CAPSULE NETWORKS FOR COMPUTER VISION
APPLICATIONS

By

Asmita Limaye	2018B5A70881G
Parth Agrawal	2018A7PS0218G
Utkarsh Singh	2018A3PS0368P
Heth Sheth	2018A7PS0110G

AT



सीएसआईआर - केन्द्रीय इलेक्ट्रॉनिकी अभियांत्रिकी अनुसंधान संस्थान
CSIR-Central Electronics Engineering Research Institute

CSIR – CEERI, Pilani

A Practice School-I station of



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

BIRLA INSTITUTE OF TECHNOLOGY AND
SCIENCE, PILANI

(June 2020)

A Report
ON
CAPSULE NETWORKS FOR COMPUTER VISION APPLICATIONS

By

Asmita Limaye	2018B5A70881G	M. Sc Physics and B.E.(Hons.) Computer Science
Parth Agrawal	2018A7PS0218G	B.E.(Hons.) Computer Science
Utkarsh Singh	2018A3PS0368P	B.E.(Hons.) Electrical and Electronics
Heth Sheth	2018A7PS0110G	B.E.(Hons.) Computer Science

Prepared in partial fulfilment of the
Practice School-I Course Nos.
BITS C221/BITS C231/BITS C241

AT



सीएसआईआर - केन्द्रीय इलेक्ट्रॉनिकी अभियांत्रिकी अनुसंधान संस्थान
CSIR-Central Electronics Engineering Research Institute

CSIR – CEERI, Pilani

A Practice School-I station of



BITS Pilani
Pilani | Dubai | Goa | Hyderabad

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

(June 2020)

ACKNOWLEDGEMENTS

We would like to extend our heartfelt gratitude to all those responsible for providing us interns with the opportunity of a lifetime to work in one of the most premier Research Institutes as CEERI, Pilani. We would like to take this opportunity to thank the several entities and personalities responsible for providing us with such a platform.

Firstly, we would like to thank Dr. D. K. Aswal, Director of CSIR-CEERI, Pilani, to have facilitated the collaboration of BITS Pilani University and CEERI and to have been willing to train interns. We are forever indebted for this learning opportunity. We would like to thank Mr. Vinod Verma, coordinator of PS program at the organization, for helping us throughout our time there.

We would like to thank Dr. Sanjay Singh, Cognitive Computing Department, for taking time out of his busy schedule to guide us and enlighten us with our project. We would like to thank Sir Sumeet Saurav, who is our mentor and has helped us with our queries on our project.

We would like to extend our gratitude to Dr. Pawan Sharma and Dr. Shishir Maheshwari, our PS instructors for their extremely insightful role in guiding us and for smoothly facilitating the collaboration. We would like to thank the PS Division of BITS Pilani University for taking such a brilliant and necessary initiative.

We would like to thank everyone else who was directly or indirectly involved to provide us this opportunity.

ABSTRACT SHEET

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI (RAJASTHAN)

Practice School Division

Station: CEERI, Pilani

Centre: Pilani

Duration: May 18 to June 28, 2020 – 6 weeks

Date of Start: May 18, 2020

Date of Submission: June 24, 2020

Title of the Project: CAPSULE NETWORKS FOR COMPUTER VISION APPLICATIONS

ID No. /Name(s)/Discipline(s) of the student(s):

Asmita Limaye	2018B5A70881G	M. Sc Physics and B.E.(Hons.) Computer Science
Parth Agrawal	2018A7PS0218G	B.E.(Hons.) Computer Science
Utkarsh Singh	2018A3PS0368P	B.E.(Hons.) Electrical and Electronics
Heth Sheth	2018A7PS0110G	B.E.(Hons.) Computer Science

Name(s) and designation(s) of the expert(s):

Dr. Sanjay Singh, Principal Scientist & Group Head, Cognitive Computing Group, Cyber Physical Systems Area

Sir Sumeet Saurav, Scientist in Cognitive Computing group.

Name(s) of the PS Faculty: Mr. Pawan Sharma. Dr. Shishir Maheshwari

Key words: Deep Learning, CNNs, CapsuleNets, Machine Learning, Neural Networks, Tensorflow, Keras.

Project Areas: Computer Vision

Abstract: CapsuleNets are a relatively new architecture in the field of computer vision. 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. They aim at combating the shortcomings of the traditional CNNs. This report highlights our plan to build a model on Kannada MNIST dataset using CapsNet.

Signatures of Students

Signature of PS Faculty

Date

Date

TABLE OF CONTENTS

1. INTRODUCTION	1
2. MOTIVATION BEHIND CAPSULE NETS	2
3. CAPSULE NETWORKS	5
4. ROUTING ALGORITHM & LOSS	9
5. PROJECT DESCRIPTION	12
6. BASIC CAPSULE NETWORK ARCHITECTURE	14
7. ARCHITECTURAL MODIFICATIONS AND NOVELTIES	15
8. EXPERIMENTS	21
9. COMPARISON BETWEEN CNN AND CAPSULENET	25
10.CONCLUSION	29
11.BIBLIOGRAPHY	30
12.REFERENCES	31
13.GLOSSARY	32

1. INTRODUCTION

CapsuleNets were proposed in a landmark paper by Geoffrey E Hinton [1] in order to combat some of the drawbacks of Convolutional Neural Networks, which throw away valuable information in their subsampling layers, and need to be trained from various viewpoints to produce accurate results. Borrowing novel concepts of pose, or position and orientation, from the field of computer graphics, CapsuleNets similarly try to model the global position and orientation of an object from the position and orientation of its local parts. This leads to less training data and an ability for neural activities to recognize the different viewpoints of an object, and has led to considerable improvements in facial recognition, among other fields.

The implementation of CapsuleNets involve vector quantities instead of the traditional scalar ones used in deep learning, which model the probability as well as the pose of constituent features. The spatial structures between the global pose of the entire object and the local pose if its parts are modelled via transformation matrices. Active capsules at one level make predictions, via transformation matrices, for the instantiation parameters of higher-level capsules. When multiple predictions agree, a higher-level capsule becomes active. An iterative routing by agreement algorithm ensures that a lower-level capsule prefers to send its output to higher level capsules whose activity vectors have a big scalar product with the prediction coming from the lower-level capsule.

In this project, we discuss the motivation behind CapsuleNets and briefly explain their theoretical background. We have implemented CapsuleNets on the Kannada MNIST database, to great results. Our goal was to compare the performance of CapsuleNets against that of a Convolutional Neural Network on this database, and identify the pros and cons of each technique to determine the effectiveness of CapsuleNets on different databases.

2. MOTIVATION BEHIND CAPSULE NETS

2.1 CONVOLUTIONAL NEURAL NETS

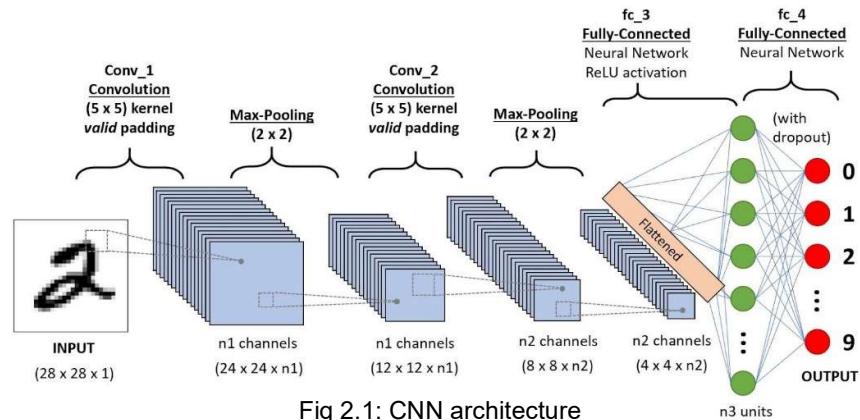


Fig 2.1: CNN architecture

Convolutional Neural Networks [4] are deep learning models where various stacks of layers perform the functions of convolution, subsampling or pooling and nonlinear operations.

The convolution operation involves layers of filters running over feature matrices and convoluting with them to get the next layer. Between feature extraction or convolutional layers, you have subsampling or pooling layers and nonlinear layers. Max pooling [5] is an example of a subsampling layer, which breaks the feature matrix down into smaller sub matrices and takes salient features from each sub matrix. Pooling reduces inputs to the next layer which lets us pinpoint more features as well as reduce computational cost.

If a feature is shifted by a small distance, the feature matrix corresponding to it will shift equally. However, after max pooling, the output would remain unaffected. If pooling was not employed, CNNs would only recognize data which was very close in viewpoint or orientation to the data set it was trained on.

This raises many problems. To recognize the object as a whole, you need the spatial relationships between components of the object. Because CNNS don't preserve spatial relationships, these are thrown away. For example, to identify a face, you have to ensure that the lips, eyes are in the correct position. But CNNs identify the features of lips and eyes individually and thus recognize any picture containing these as a face.

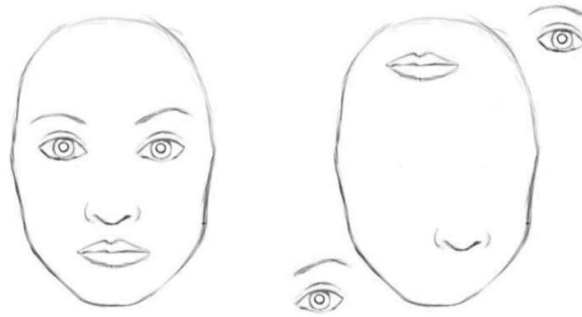


Fig 2.2: Features of face arranged randomly

2.2 INVARIANCE AND EQUIVARIANCE

An operator is equivariant with respect to a transformation when the effect of the transformation is detectable in the operator output. An operator is invariant with respect to a transformation when the effect of the transformation is not detectable in the operator output.

Subsampling manages to add translational invariance at each level, because the pooling takes the maximum or average over a small area. The label is view invariant, so the actual location of the image doesn't matter as much. CNNs try to make the neural activities invariant to small changes in viewpoints, because the labels are invariant.

However, we should aim at having neural activities equivariant, or able to identify different viewpoints. In perceptual systems, like the human brain, the weights have viewpoint invariant knowledge but the neural activities change depending on the orientation of the object.

2.3 INVERSE COMPUTER GRAPHICS

In computer graphics, the combination of position and orientation with respect to a coordinate system is referred to as the pose of an object. In order to obtain a particulate view of a part, the pose of the whole object or global pose is multiplied by a transformation matrix to obtain a local pose of a part of the object from a particular viewpoint.

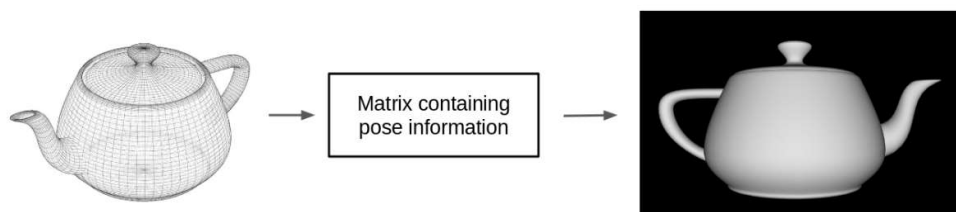


Fig 2.3: Computer graphics

This is achieved through a hierarchical model on which the relationship of the spatial structure of the component parts of the object and the whole object is modelled via matrices which are viewpoint invariant and preserve relationships. We can change the position and orientation of a part of the object and get back the position and orientation of the whole object using the matrices. Our aim here is to model higher levels of a vision system like the model followed in computer graphics, to go from a representation of a part of an object to a representation of the whole object via viewpoint invariant matrix weights which have global pose information

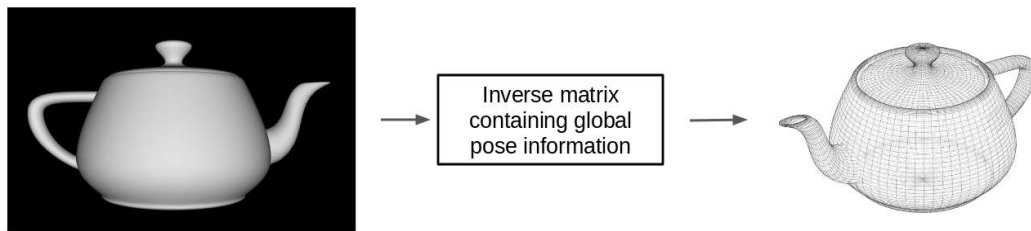


Fig 2.4: Inverse Computer graphics

2.4 PROS AND CONS

2.4.1 PROS

- Great results have been obtained for the purposes of image classification, segmentation and object detection
- High recognition accuracy has been obtained on MNIST dataset, and promising results have been exhibited on the CIFAR10 database
- The training dataset required for Capsule Networks is of a much smaller size

2.4.2 CONS

- The expensive operation of the inner loop in the routing by agreement algorithm leads to a slow learning process.
- There is a currently a lack of testing, which thus concrete conclusions about its effectiveness in the future cannot be drawn
- The crowding effect leads to the network not being able to identify similar looking objects which are situated close to each other.
- The computing and implementation complexity is more when compared to traditional CNNs.

3. CAPSULE NETWORKS

3.1 Basic structure of a Capsule:

As defined in the paper Dynamic Routing Between Capsules[1], 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. Capsules perform a lot of internal computation and then encapsulate the results of these computations into a n-dimensional vector and output this vector. This becomes the fundamental point of difference between a neuron and a capsule.

A capsule takes in vector inputs and outputs vector quantities whereas a neuron takes in scalar inputs and outputs scalar quantities and it is this difference that lends a huge advantage to capsules over neurons since one can encapsulate far greater amounts of information in a vector quantity than in a scalar.

Just as a bunch of neurons in each layer of a traditional CNN are responsible for detecting entities in a given perceptual field, a capsule is responsible for detecting an entity in its perceptual field.

The 2 main pieces of information embedded in the vector output of a capsule are:

1. The probability that a particular entity exists.
2. The instantiation parameters of the entity.

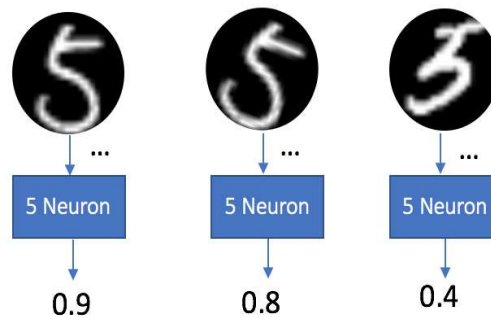
The probability of the existence of an entity is embedded in the length of the output vector whereas the instantiation parameters are embedded in the vector's orientation.

3.2 Instantiation parameters:

The orientation of the vector in n-dimensional space encodes the instantiation parameters of the entity. The parameters could be anything from the shape and size of the object to the hue, texture, albedo, etc. CNNs, as pointed out by Hinton, et al., are viewpoint invariant, in that if the viewpoint of the input image changes a little, the activities of the neurons responsible for detecting a particular entity/object remain the same. This invariance of neuronal activities is what the novel architecture of Capsule Networks strives to overcome with equivariance of activities. This will allow the network to detect an object/entity/feature regardless of the viewpoint of the input image. The output vectors are equivariant with regards to the instantiation parameters and invariant to

the probability of detecting a particular entity.

For example: let there be 3 neurons, each being fed the number '5' as the input image. The scalar nature of the output of a neuron restricts it to outputting only the confidence in it having detected the number correctly.



Now instead of a neuron, let the input be fed to a capsule.

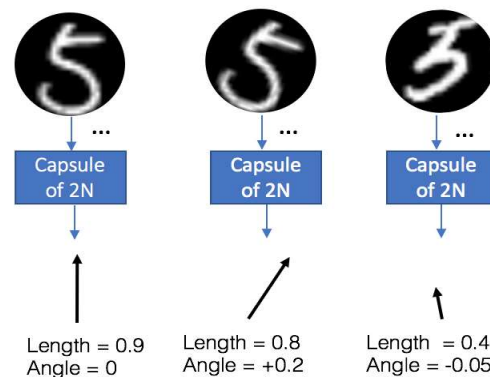


Figure 3.1: Input image fed to a conventional neuron v/s Input image fed to a capsule.

Capsules also output the same confidence as the neurons but the confidence is embedded in the length of the output vectors (also referred to as activation vectors in popular literature), and the angle the input image makes with an imaginary axis is embedded in the direction of the vectors. Here the angle is an instantiation parameter and the length of the vector is the probability/confidence of the capsule detecting the number 5.

Imagine that a capsule detects a face in the image and outputs a 3D vector of length 0.99. Then we start moving the face across the image. The vector will rotate in its space, representing the changing state of the detected face, but its length will remain fixed, because the capsule is still sure it has detected a face. This is what Hinton refers to as activities equivariance:

neuronal activities will change when an object “moves over the manifold of possible appearances” in the picture. At the same time, the probabilities of detection remain constant, which is the form of invariance that we should aim at, and not the type offered by CNNs with max pooling.

3.3 What do the individual dimensions of the output vector of a capsule represent:

Since we pass the encoding of only 1-digit every time we feed an input image, the dimensions of the output vector of a capsule should learn to span all the possible variations in the way a digit can be represented/instantiated. The variations include ones that are general to all the digit classes and ones that are specific to a digit like the size of the loop in the number ‘6’.

After successfully computing the activity vector of a digit, we can feed perturbed versions of this activity vector to the reconstruction network (particulars of which have been covered in the coming pages) to see how one particular dimension affects the representation of the image.






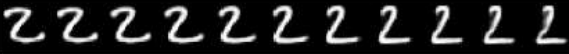
Scale and thickness	
Localized part	
Stroke thickness	
Localized skew	
Width and translation	
Localized part	

Figure 3.2: Dimension perturbations, each row shows the reconstruction when one of the 16 dimensions of the output of the Digit-Caps layer is perturbed. [1]

3.4 How is the output of a capsule generated:

3.4.1 Input to the capsule:

the input to a capsule is a vector coming from either a lower layer of capsules or from the output of a traditional convolutional layer. Any vector output from a lower capsule layer (“child” layer) is sent to all possible capsules in the next higher (or “parent”) layer.

3.4.2 Affine transformation of input vectors (matrix multiplication by weight/transformation matrices):

The input vectors are multiplied by affine transformation matrices to produce the prediction of every capsule of how a particular higher-level capsule’s output should look like. The

transformation matrices are those “assigned” by the higher-level capsule to all of the lower level capsules which encode the spatial relationship between the higher-level feature (say, a face) and the lower level feature (mouth, nose, etc). These transformed inputs or predictions as they are referred to in popular literature, are then accumulated by weighted sums, the weights or coefficients of which are decided in the Dynamic Routing step, the algorithm at the heart of capsule networks’ functioning.

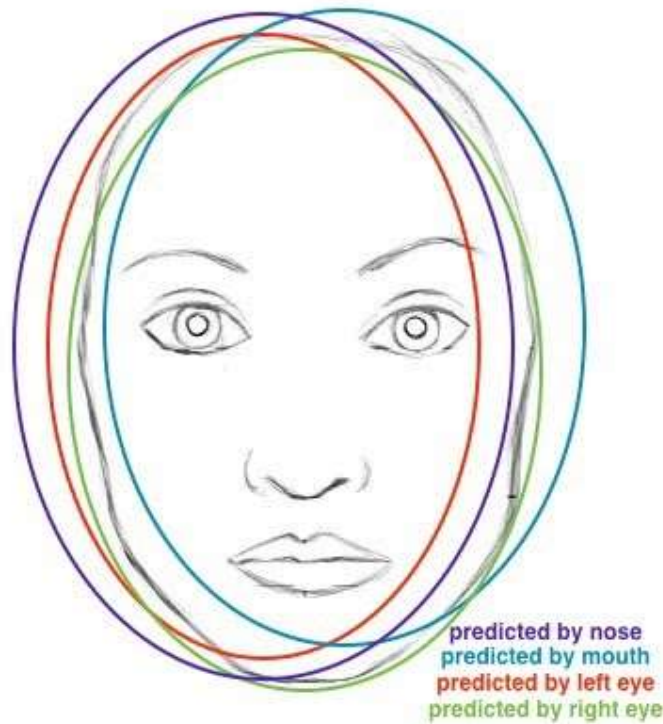


Figure 3.3: shows the predictions of 4 lower level features of a higher-level feature(face).

Here the predictions seem to match, indicating the presence of a face in the image.

This agreement/disagreement between the predictions of lower level capsules is taken care of in the Dynamic Routing algorithm.

$$\hat{\mathbf{u}}_{j|i} = \mathbf{W}_{ij} \mathbf{u}_i$$

Figure 3.4: \mathbf{W}_{ij} is the transformation matrix for higher level capsule ‘j’ and lower level capsule ‘i’, \mathbf{u}_i is the input vector to the lower level capsule i and the LHS represents ‘i’s’ prediction of ‘j’.

4. ROUTING ALGORITHM & LOSS

According to Hinton [1], when a visual stimulus is triggered, the brain has an inbuilt mechanism to “route” low level visual data to parts of the brain where it believes can handle it best. CNNs perform this process via pooling, a very primitive way to do it as it only transfers the most active part of a neuron. CapsNet is different as it tries to send the information to the capsule above it that is best at dealing with it. There are many possible routing algorithms to implement CapsNet. The algorithm proposed in the main paper [1] is the “Dynamic Routing algorithm”.

4.1 DYNAMIC ROUTING BETWEEN CAPSULES

The vector output of a capsule layer makes it possible to use a powerful dynamic routing algorithm. The essence of this algorithm is that a lower level capsule(i) will send its input to the higher-level capsule(j) that “agrees” with its input. First, for every predicted output (derived after the affine transformation), we start by setting a raw routing weight b_{ij} equal to 0.

Procedure 1 Routing algorithm.

```

1: procedure ROUTING( $\hat{u}_{j|i}$ ,  $r$ ,  $l$ )
2:   for all capsule  $i$  in layer  $l$  and capsule  $j$  in layer  $(l + 1)$ :  $b_{ij} \leftarrow 0$ .
3:   for  $r$  iterations do
4:     for all capsule  $i$  in layer  $l$ :  $c_i \leftarrow \text{softmax}(b_i)$  ▷ softmax computes Eq. 3
5:     for all capsule  $j$  in layer  $(l + 1)$ :  $s_j \leftarrow \sum_i c_{ij} \hat{u}_{j|i}$ 
6:     for all capsule  $j$  in layer  $(l + 1)$ :  $v_j \leftarrow \text{squash}(s_j)$  ▷ squash computes Eq. 1
7:     for all capsule  $i$  in layer  $l$  and capsule  $j$  in layer  $(l + 1)$ :  $b_{ij} \leftarrow b_{ij} + \hat{u}_{j|i} \cdot v_j$ 
   return  $v_j$ 

```

Algorithm 1: Routing Algorithm

The next part of the algorithm is to iteratively update the weights by performing 4 operations. First, we need to ensure that the weights c_{ij} are probabilistic in nature i.e. they are positive and add up to 1. For this, the softmax function is applied on the raw weights, for each primary capsule.

$$c_{ij} = \frac{\exp(b_{ij})}{\sum_k \exp(b_{ik})}$$

In the 1st iteration, the output is sent to all possible higher capsules but is scaled down by coupling coefficients that sum to 1. In the 2nd step, the total input to a capsule s_j is calculated as the weighted sum of all “prediction vectors” $u_{j|i}$ with weights c_{ij} .

$$s_j = \sum_i c_{ij} \hat{u}_{j|i} ,$$

The lengths of the output vectors need to be scaled down to less than 1. In order to ensure this, we use a non-linear “squashing” function which preserves the direction but reduces the length.

$$\mathbf{v}_j = \frac{\|\mathbf{s}_j\|^2}{1 + \|\mathbf{s}_j\|^2} \frac{\mathbf{s}_j}{\|\mathbf{s}_j\|}$$

The smaller vectors are reduced to nearly zero while the longer ones are closer to 1. Therefore, the likelihood of each capsule is bounded between zero and one.

$$\begin{aligned} v_j &\approx \|\mathbf{s}_j\| s_j && \text{for } s_j \text{ is short} \\ v_j &\approx \frac{s_j}{\|\mathbf{s}_j\|} && \text{for } s_j \text{ is long} \end{aligned}$$

So far, we have calculated the outputs of the outer layer capsules. The next step is where the weight update happens which is the essence of this algorithm. In this step, we calculate the agreement between the current output \mathbf{v}_j of each capsule, j , in the parent layer and the prediction $\mathbf{u}_{j|i}$ made by capsule i . This is done by taking their dot products and adding it to the initial weight so that those pairs with higher similarity have their weights increased while those that do not agree have their weights decreased.

$$b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i} \cdot \mathbf{v}_j$$

This process is repeated for r iterations after which the output vector of the higher capsule \mathbf{v}_j is returned. This algorithm is quite different from how CNNs work. The differences are summarized in the table below:

Capsule vs. Traditional Neuron			
Input from low-level capsule/neuron		vector(\mathbf{u}_i)	scalar(x_i)
Operation	Affine Transform	$\hat{\mathbf{u}}_{j i} = \mathbf{W}_{ij} \mathbf{u}_i$	—
	Weighting	$\mathbf{s}_j = \sum_i c_{ij} \hat{\mathbf{u}}_{j i}$	$a_j = \sum_i w_i x_i + b$
	Sum		
	Nonlinear Activation	$\mathbf{v}_j = \frac{\ \mathbf{s}_j\ ^2}{1 + \ \mathbf{s}_j\ ^2} \frac{\mathbf{s}_j}{\ \mathbf{s}_j\ }$	$h_j = f(a_j)$
Output		vector(\mathbf{v}_j)	scalar(h_j)

Figure 4.1: CapsNet vs Traditional Neuron

4.2 LOSS FUNCTION

4.2.1 MARGIN LOSS

A separate margin loss L_c [1] is used for each category of digit capsules. It is given by the formula:

$$L_c = T_c \max(0, m^+ - \|v_c\|)^2 + \lambda(1 - T_c) \max(0, \|v_c\| - m^-)^2$$

where $T_c = 1$ if a digit of that class is present, and $m^+ = 0.9$ and $m^- = 0.1$. Therefore, if an object of class c is present, then $\|v_c\|$ should be no less than 0.9. If not then $\|v_c\|$ should be no more than 0.1. The λ down-weighting of the loss for absent digit classes stops the initial learning from shrinking the lengths of the activity vectors of all the digit capsules. $\lambda = 0.5$ is used. The total loss is given by the sum of losses over all digit capsules.

4.2.2 RECONSTRUCTION AS REGULARIZATION

In addition to the Capsule network, a decoder network is used to reconstruct the original image. For this, everything except the activity vector of the correct digit capsule is masked. Then, this activity vector is used to reconstruct the image. Usually 3 fully connected layers along with Sigmoid activation is used for this purpose. After that the sum of squared errors is minimized between the reconstructed image and the original image.

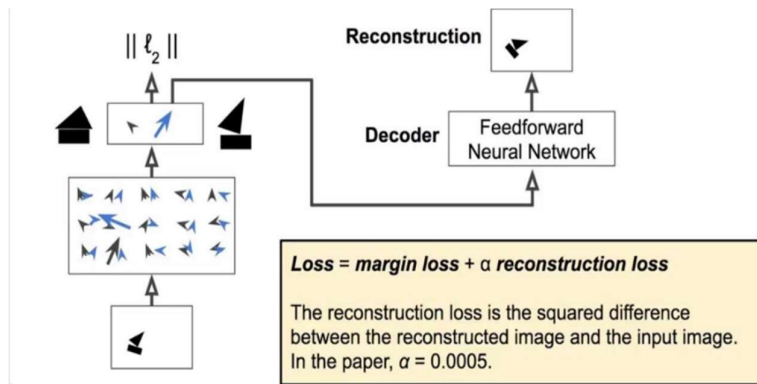


Fig 4.2: Reconstruction

The decoder ensures that the network preserves all the information that is needed to reconstruct the image up to the top layer. This has a regularizing effect and prevents overfitting. It also helps in generalizing the network for other datasets. So, the overall loss is given by:

Loss = margin loss + α * reconstruction loss [1]

5. PROJECT DESCRIPTION

In this project, our aim is to provide practical support to the theoretical claim that CapsuleNets are better than Convolutional Neural Networks. For this purpose, we shall be implementing the CapsuleNets on the Kannada MNIST Dataset. We shall try different layers and routing mechanisms to improve the accuracy of the Capsule Networks. After that, the results will be compared on the basis of recognition accuracy, computational time and memory usage. Further, review of some literature based on CapsuleNets will be done to support the claim.

5.1 DATASET USED

5.1.1 MNIST [2]

It consists of 70000 28x28 black and white images which have been distributed in 10 classes (digits 0-9), with 7000 images per class. It is divided into 60000 training images and 10000 test images.



Figure 5.1: MNIST database

5.1.2 KANNADA MNIST [3]

The main Kannada-MNIST dataset contains a training set of 60000 28×28 gray-scale sample images and a test set of 10000 sample images uniformly distributed across the 10 classes. The Dig-MNIST Dataset is an additional test set for the Kannada MNIST Dataset with difficult cases. In particular, the digits 3 and 7 cause some problems due to their resemblance to Arabic digit 2. Further, the digit 6 is written in multiple ways which makes it harder for the network to recognize it, lowering the accuracy.

೦	೧	೨	೩	೪	೫	೬	೭	೮	೯	೦೦
ಒಂದು	ಎರಡು	ಮೂರು	ನಾಲ್ಕು	ಐದು	ಆರು	ಏಳು	ಎಂಟು	ಒಂಬತ್ತು	ಹತ್ತು	
omdu	eraḍu	mūru	nāḷku	aidu	āru	ēḷu	eṁṭu	ombattu	hattu	
1	2	3	4	5	6	7	8	9	10	

Figure 5.2: Kannada MNIST database

5.2 TIMELINE

In the first week, we got used to the Canvas platform. We interacted with the different mentors from CSIR-CEERI, Pilani and discovered the various projects being carried out there. We also realized about the real-life impact that these projects have. After that, we chose a project of our choice. We also learned TensorFlow and Keras which would be used later in our project.

We utilized that week to complete the Deep Learning Specialization by Andrew Ng on Coursera. In the specialization, we were introduced to Convolutional Neural Networks and their use of filters to capture the spatial and temporal dependencies in an image.

We also explored the implementation of CapsNet on CIFAR10 given in the Keras documentation. After that, we built a basic CapsuleNets model (discussed later) and tested its accuracy on the MNIST dataset. The model on the MNIST dataset was improved by trying out different layers and activations. We also tried to find issues with the incumbent routing algorithms.

In the next week, we implemented this CapsuleNet model on the Kannada MNIST Dataset.

We shall try to improve the accuracy of the CapsuleNets using different deep learning techniques on the Kannada MNIST Dataset. We used techniques like data augmentation and ensemble to boost our model's performance.

In the next week, we built a CNN model (discussed later) with a comparable number of parameters and compared it with the Kannada-MNIST on the basis of recognition accuracy and computation time. The results of the same have been discussed later in the report.

The research can be continued further by trying CapsuleNets on other datasets, modifying the architecture of CapsuleNets.

6. BASIC CAPSULE NETWORK ARCHITECTURE

The network, as proposed in the paper [1], has 2 convolutional layers followed by a fully-connected layer.

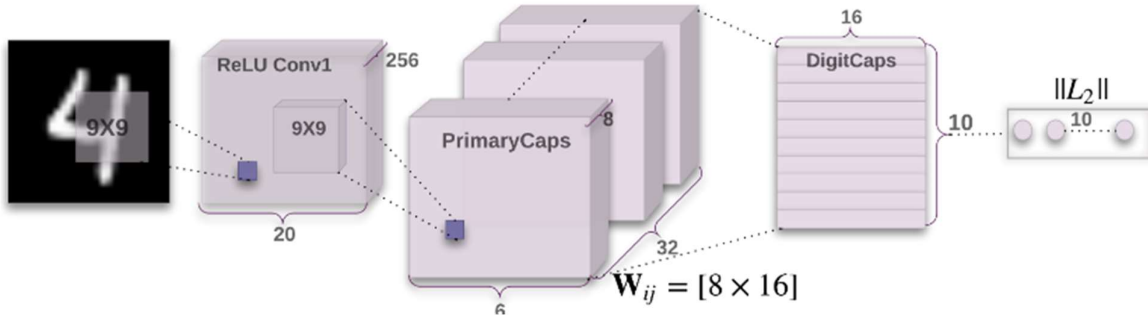


Figure 6.1: Basic CapsNet model

The input image is fed as a NumPy array of (28, 28, 1) which after ReLU Conv1(kernel size = 9, number of kernels = 256, stride = 1) and a Batch Normalization layer becomes (20, 20, 256). This is then fed to another convolutional layer [4], with the same number of kernels and kernel size as the first but with a stride of 2, hence the output of that layer is a (6, 6, 256) NumPy array. Before feeding this output to the dynamic routing algorithm, it is reshaped(flattened) to (1152, 8).

This (1152, 8) output can be seen as 1152 vectors each of 8 dimensions, i.e., 1152 primary capsules each having a dimensionality of 8. The dynamic routing takes place between these capsules and the 10 16-dimensional capsules in the digit caps layer. The final prediction of the network is the vector with the largest magnitude (longest vector).

What follows the digit caps layer is the decoder network, which again, as proposed in the paper[1] is a FCN with a sigmoid activation at the end, where the last dense layer has 784 units which are then reshaped into (28, 28, 1) array, the size of the input image fed.

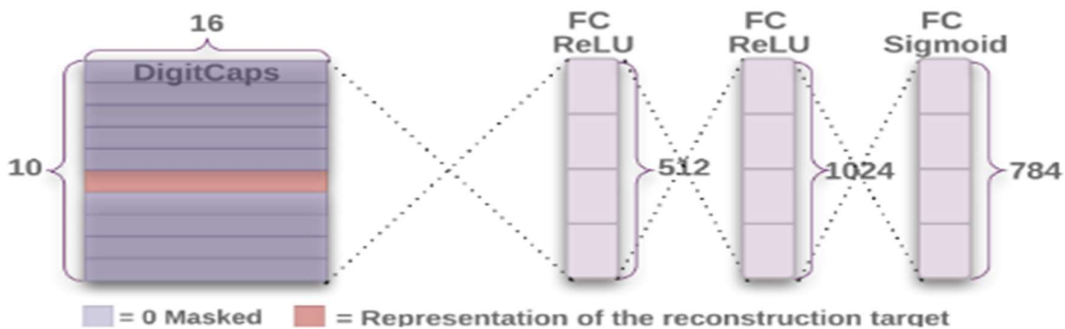


Figure 6.2: FCN(Dense) as a decoder/reconstruction network

We trained this basic CapsNet on the Kannada MNIST [3] and DIG-10K dataset and were able to achieve **97.84%** test accuracy and **81.12%** DIG-10K set accuracy after just 12 epochs. This was already well beyond the baseline accuracy reported in [3].

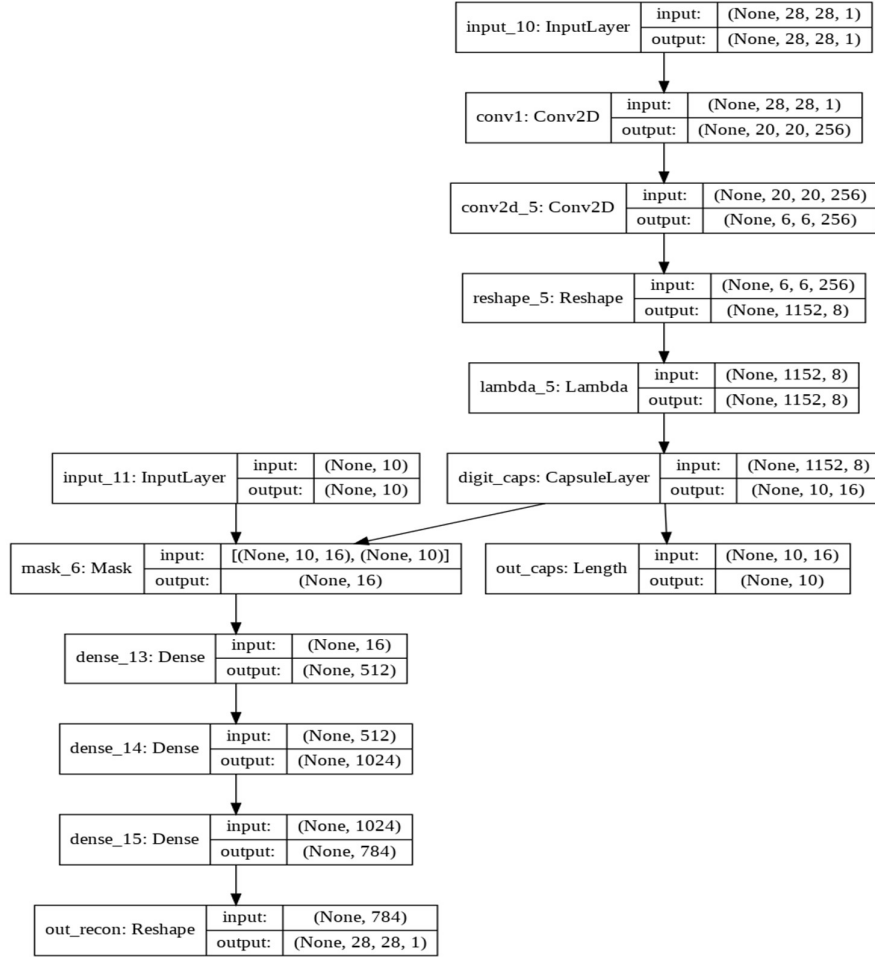


Figure 6.3: Basic CapsNet architecture. This model architecture is from the original paper [1].

6.1 CapsNet on MNIST dataset

A basic implementation of a CapsNet [1] was trained on the MNIST [2] digits dataset to understand its functioning. The model, shown in figure 6.3, consisted of an input layer, followed by two convolutional layers. After that, the result was squashed and passed through the Capsule Layer. The decoder which was used to reconstruct the image, consists of three Dense layers. This basic architecture was able to achieve a 99.35% accuracy on the test set after just 10 epochs. Such promising results meant that we were only able to improve upon them from here on our main task; the implementation of the CapsNet architecture to the Kannada MNIST dataset.

7. ARCHITECTURAL MODIFICATIONS AND NOVELTIES

Many modifications, some small, some big were tried on the basic architecture to eke out every possible ounce of performance from it. The 3 major ones were:

- Increasing the depth of the convolutional stack preceding the primary capsule layer
- Increasing the dimensionality of the digit capsules
- Transposed convolutional network of layers as the decoder/reconstruction network

7.1 INCREASING THE DEPTH OF THE CONVOLUTIONAL STACK

Instead of having one convolutional layer before feeding its output to the Primary capsule layer, we had 2 of them. This meant that the actual **Capsule layers** [1] had less amount of *work* to do because the Primary Capsule layer was being fed more complex features from the Convolutional layers before it rather than having to extract those itself. This makes intuitive sense because it is easier to piece together a whole from larger and a fewer number of parts than when compared to doing the same task using smaller and a larger number of parts.

Increasing the depth of the conv stack did just that, it already learned complex shapes for the Primary Capsule layer to then route to the Digit Capsule layer, hence making the routing task a little easier and hence taking the impetus of learning away from the capsule layers and giving it to the convolutional layers instead.

This was a major reason that this particular modification was pursued further since this project's main objective was to understand the functioning of the CapsNet and leverage it for solving computer vision problems. This was also reinforced by the slight decrease in the performance (accuracy wise) on the Kannada MNIST dataset.

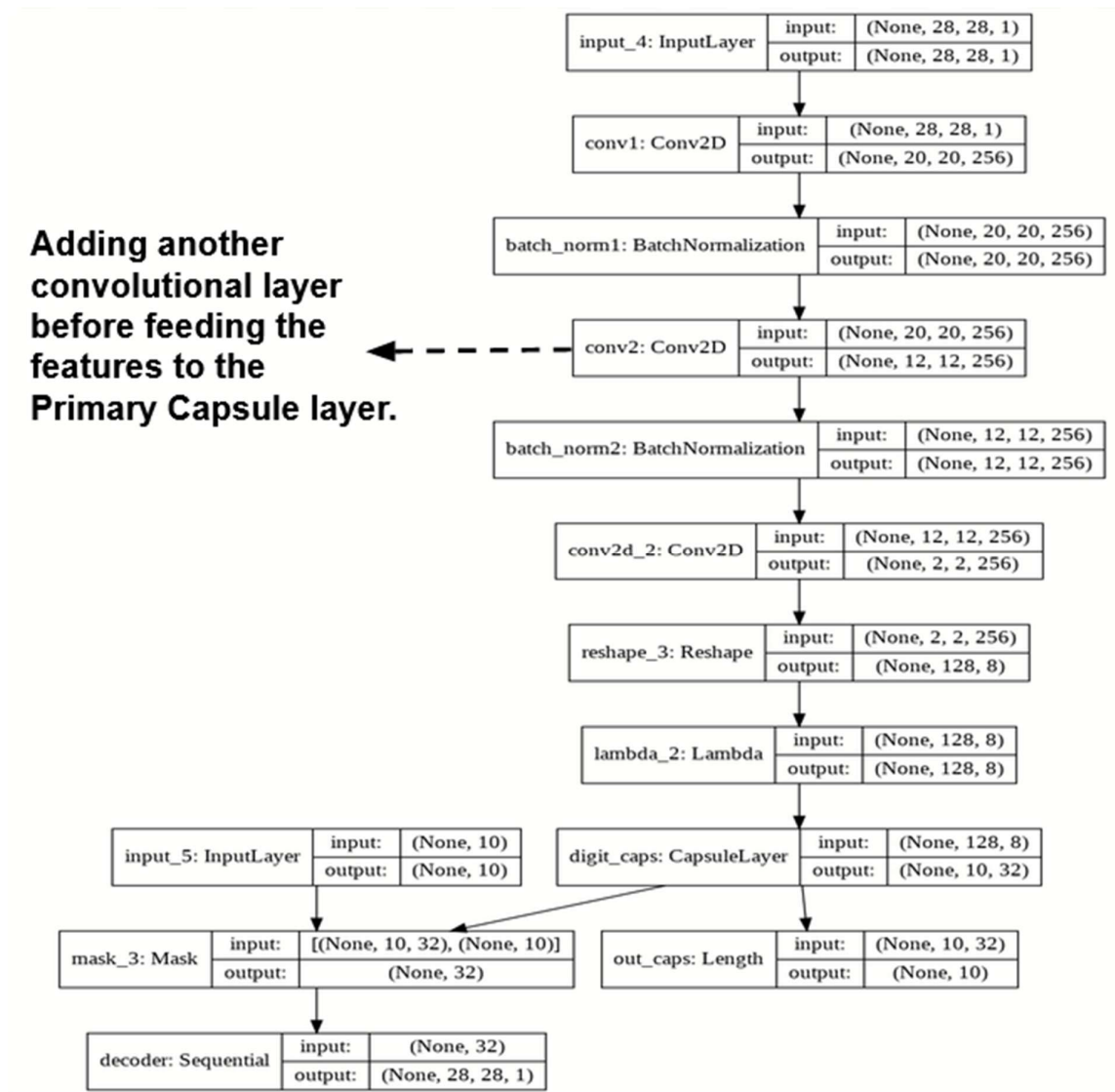


Figure 7.1: CapsNet architecture with increased depth of the convolutional stack preceding the Primary Capsule layer.

7.2 INCREASING THE DIMENSIONALITY OF THE DIGIT CAPSULES

Following from the comments made in the previous section about wanting to lay most of the stress of learning on the Capsule layers, this change to the architecture follows intuitively.

The original paper [1] states that the Digit Capsule layer have capsules/vectors each of 16 dimensions; we increased it to 32.

This change to the architecture would mean a significant increase in the number of trainable parameters and in the training time, but the change that follows more than makes up for it, at least by the way of trainable parameters.

Since the number of trainable parameters increased due to this change, it reached a lower accuracy after 12 epochs than the 97.84% of the basic CapsNet architecture but the training process was much less noisy and the plots of the loss and accuracy metrics had a much better trend than the basic model.

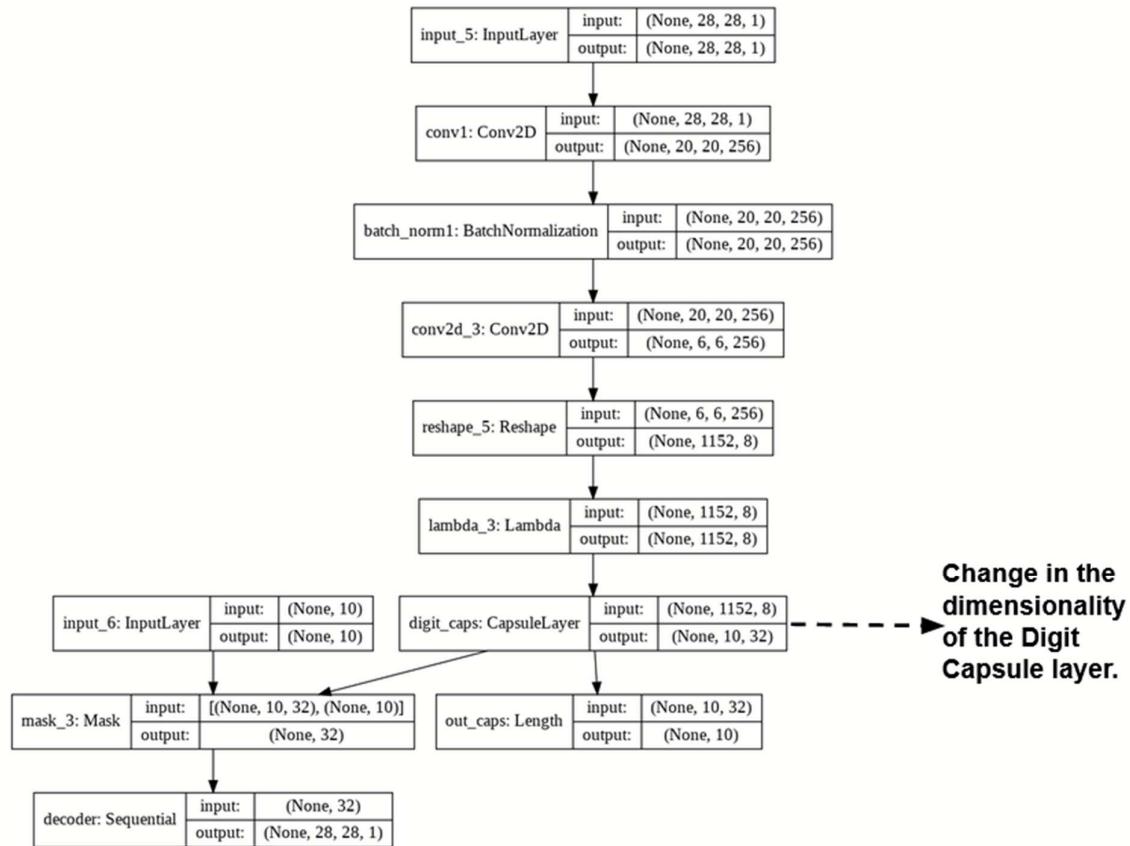


Figure 7.2: Increasing the dimensionality of the digit capsules

This is also the final encoder model (Capsule network) we used.

7.3 USING A TRANSPOSED CONVOLUTIONAL DECODER NETWORK

In all the modifications made so far, the one major pain point with respect to the performance of the model on accuracy and loss metrics was the reconstruction loss or decoder loss, which was too high for our liking. Hence upon looking for alternative models for a decoder network, we finally zeroed in on a network of primarily transposed convolutional layers (often falsely called the deconvolutional layers, but in signal and image processing a deconvolution is a very different operation from a transposed convolution, and that particular mathematical operation hasn't gathered much traction in popular deep learning literature and hence this name is a misnomer).

The intuition of this change was drawn from the fact that most of the computer vision tasks that require a decoder network (image segmentation, instance segmentation etc.) use either the Up-sampling layers or the Transposed convolutional layers to reconstruct the image back from a set of learned features/representations.

This network takes the input from the digit capsule layer, passes them through a Dense layer and then reshapes them to (7, 7, 32) dimensional output which is serially passed through transposed convolutional layers and finally reshaped to give back the reconstructed input.

The reduction in the number of trainable parameters and the training time due to this change counters the increase in same due to the previous change (increased dimensionality of the digit capsules) such that the total number of trainable parameters remain roughly the same as the basic CapsNet model.

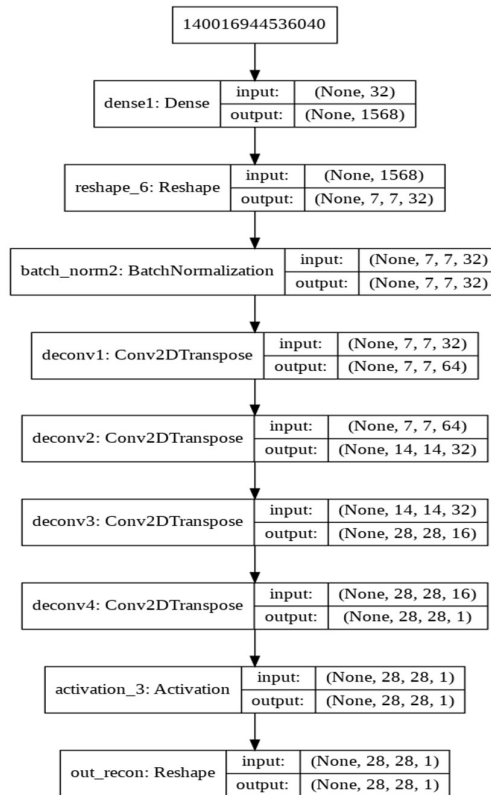


Figure 7.3: A network of transposed convolutional layers as the decoder/reconstruction network
This is the final decoder network we used.

Our model also uses a BatchNorm layer.			
Layer (type)	Output Shape	Param #	Connected to
input_4 (InputLayer)	(None, 28, 28, 1)	0	
conv1 (Conv2D)	(None, 20, 20, 256)	20992	input_4[0][0]
batch_norm1 (BatchNormalization)	(None, 20, 20, 256)	1024	conv1[0][0]
conv2d_2 (Conv2D)	(None, 6, 6, 256)	5308672	batch_norm1[0][0]
reshape_3 (Reshape)	(None, 1152, 8)	0	conv2d_2[0][0]
lambda_2 (Lambda)	(None, 1152, 8)	0	reshape_3[0][0]
digit_caps (CapsuleLayer)	(None, 10, 32)	2960640	lambda_2[0][0]
input_5 (InputLayer)	(None, 10)	0	
mask_3 (Mask)	(None, 32)	0	digit_caps[0][0] input_5[0][0]
out_caps (Length)	(None, 10)	0	digit_caps[0][0]
decoder (Sequential)	(None, 28, 28, 1)	93601	mask_3[0][0]
Total params: 8,384,929			
Trainable params: 8,372,833			
Non-trainable params: 12,096			

Our decoder network has roughly 93K trainable params as opposed to the 1.4M params the traditional Dense network(FCN) had.

Figure 7.4: Final model summary of our custom CapsNet architecture, note that our model has roughly the same number of trainable parameters as the one proposed in the paper [1].

8. EXPERIMENTS

8.1 MODEL IMPLEMENTATION

To build the model described earlier, we used python packages Keras (2.2) and Tensorflow (1.15). Due to lack of inbuilt GPUs, we used Google Colab to train and test our models. To compare our model, we implemented a standard CNN as baseline (further discussed later) and referred to other baseline models used in Jiang et. al [6]. We also used an ensemble of 5 models to try and increase the accuracy.

8.2 DATA AUGMENTATION

To further improve the model's accuracy, we use data augmentation. Benefits of data augmentation include adding data diversity and avoiding overfitting. The DIG set has images that are randomly zoomed-in/out, cropped at certain places and all sorts of such variations. The image data augmentation process helps the model become robust to some of those changes.

- **Width and Height Shift** – Images are shifted by 20% of total width and height.
- **ZCA Whitening** – To reduce the redundancy in the matrix of pixel images. Less redundancy in the image is intended to better highlight the structures and features in the image to the learning algorithm.
- **Rotation range** – The original images are randomly rotated between -20 to 20 degrees.
- **Zoom range** – The original images are randomly zoomed in and out by 20%.
- **Shear range** – A shear transformation of 0.2 is applied to the original images.

8.3 TRAINING

To choose the best model, we further divide the training set into training and validation sets with a ratio of 90% : 10%. During the training process, we use a batch size of 128 and run each model for 50 epochs. Initially we used the Adam [7] optimizer. But later we changed to the Nadam [8] optimizer with a learning rate of 0.001 which led to better convergence and slightly lower loss. Then the model weight with the best validation accuracy is saved and evaluated in the testing set.

Data Augmentation	Training accuracy (loss)	Validation Accuracy (loss)	Test Set accuracy	DIG set accuracy
No (30 epochs)	100% (0.0031)	99.43% (0.0103)	97.98%	81.73%
Yes (50 epochs)	99.58% (0.0096)	99.65% (0.0116)	98.52%	91.09%

Table 8.1: Comparison between models with and without data augmentation

8.4 EVALUATION

We use accuracy as our main evaluation metric. For the evaluation of Dig-MNIST, we use the best model trained and validated from Kannada-MNIST directly without any further revision. Table 8.1 shows our results with and without data augmentation. As predicted, due to the random zooming and cropping in the DIG images, data augmentation significantly increased the DIG accuracy. Table 8.2 compares our results with those provided in Jiang et. al [6]

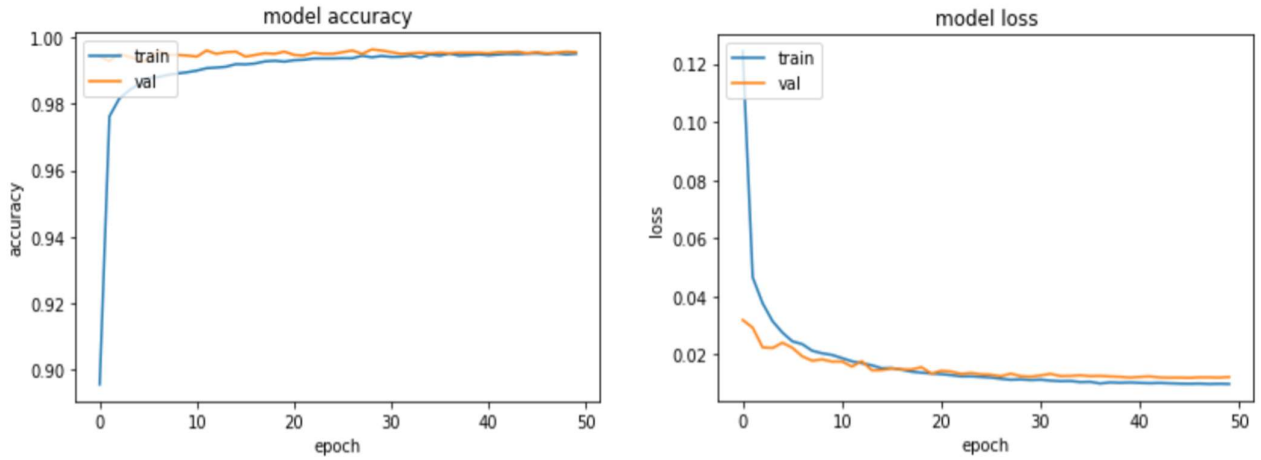


Fig 8.1: Model accuracies and losses.

Figure 8.1 shows the graphs of training and validation accuracies and losses. It is visible that both the validation and training accuracies have converged. An example of the reconstructed images is shown in figure 8.2.

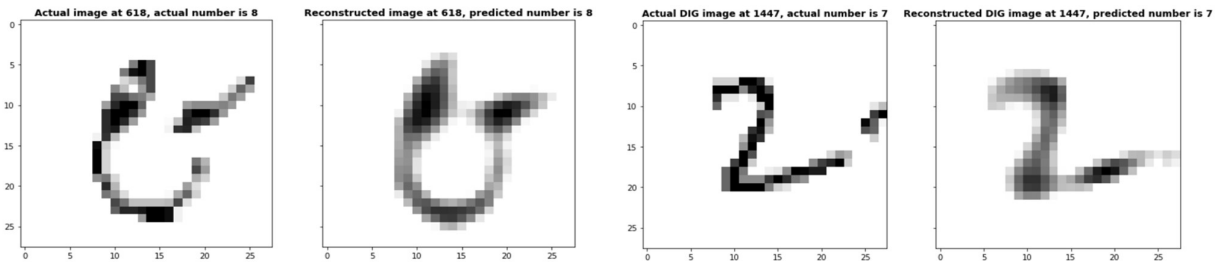


Fig 8.2: Original vs Reconstructed Images

Method \ Data		Kannada-MNIST	Dig-MNIST
Without data augmentation	CapsuleNet	97.98	81.73
	Edge-SiamNet	97.37	77.14
	Edge-TripleNet	97.21	77.10
	EdgeNet	97.34	76.59
	CNN	97.42	78.88
	LeNet	97.36	75.28
	MobileNet	97.22	79.96
	VGG16	97.76	81.20
	AlexNet	97.34	75.63
	ResNet50	97.83	78.85
With data augmentation	CapsuleNet	98.52	91.09
	CapsuleNet (Ensemble)	98.63	91.21
	Edge-SiamNet	98.24	85.50
	Edge-TripleNet	97.97	85.97
	EdgeNet	98.13	85.95
	CNN	97.44	82.96
	LeNet	97.35	80.83
	MobileNet	97.80	85.09
	VGG16	98.17	87.45
	AlexNet	97.48	84.28
	ResNet50	98.01	87.32

Table 8.2: Comparison with previous results in [6]

The images shown in figure 8.3 show that there is a confusion between 0 and 1, 5 and 4, 6 and 9. Zero is misclassified as a 1 in many cases due to the lower part being cropped. Similarly, 5 is classified as 4 due to the images being rotated and features in these images being slightly different from the usual 5. The similarity between 6 and 9 causes 6 to be misclassified as a 9.

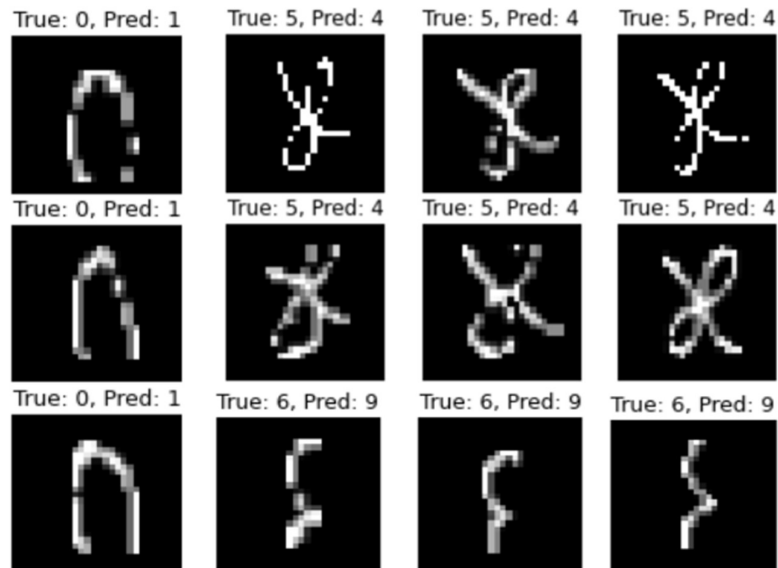


Fig 8.3: Misclassified images

This is further justified by the Confusion matrix in figure 8.4 which indicates the accuracies of individual digits. As mentioned before the DIG dataset is quite different from the train and test dataset which makes it harder to classify using the train dataset. This is visible from its confusion matrix as many digits are classified wrongly.

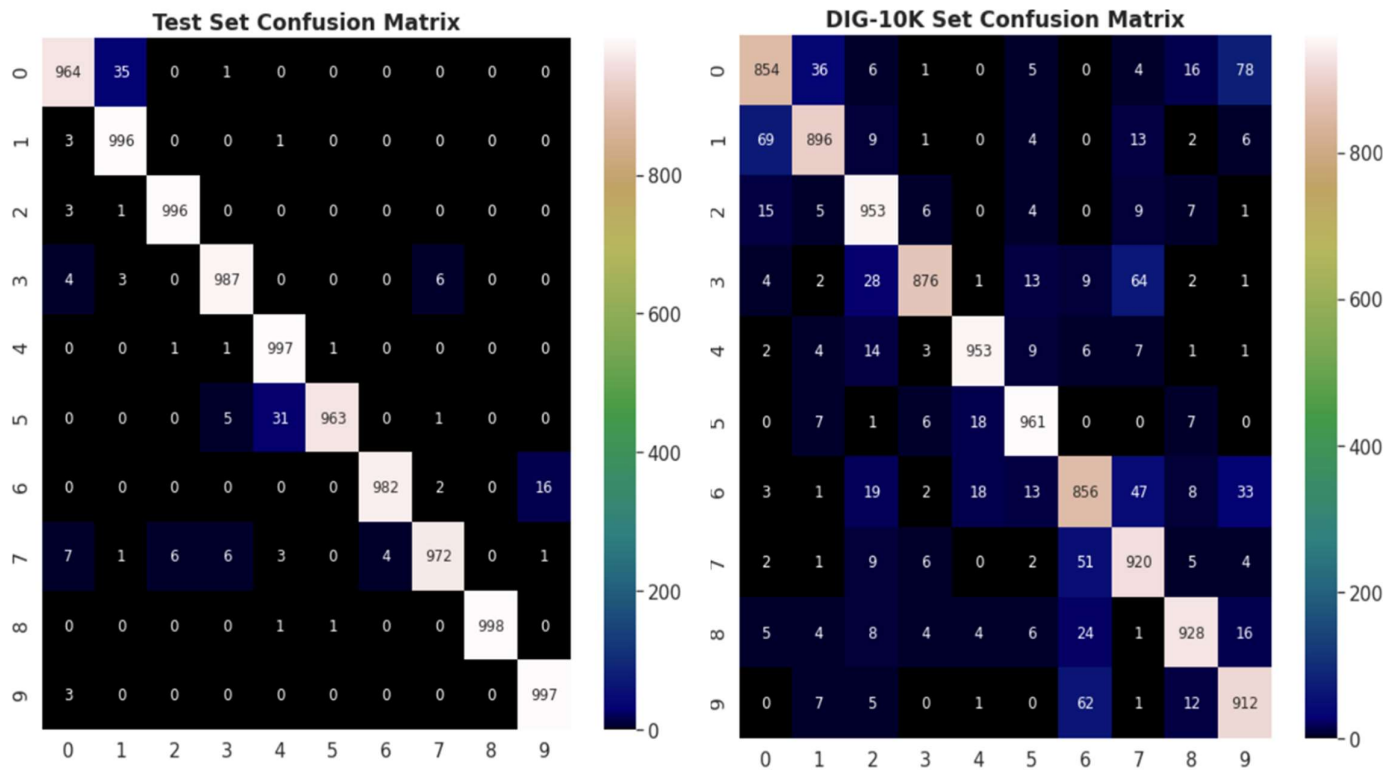


Fig 8.4: Confusion Matrix

9. COMPARISON BETWEEN CNN AND CAPSULENET

9.1 CNN MODEL USED

In order to understand the drawbacks and benefits of CapsuleNets, we decided to compare and contrast it with a CNN model on the same Kannada MNIST database. To maintain a standard for comparison, we decided to train it with the same hyperparameters= with the same learning rate, batch size, number of epochs, loss calculation, optimizer and so on. We decided to run both on Google Colab, in order to maintain the same computing power and compare the times taken per epoch by both. The Convolutional Neural Network model we arrived at after experimentation was very deep.

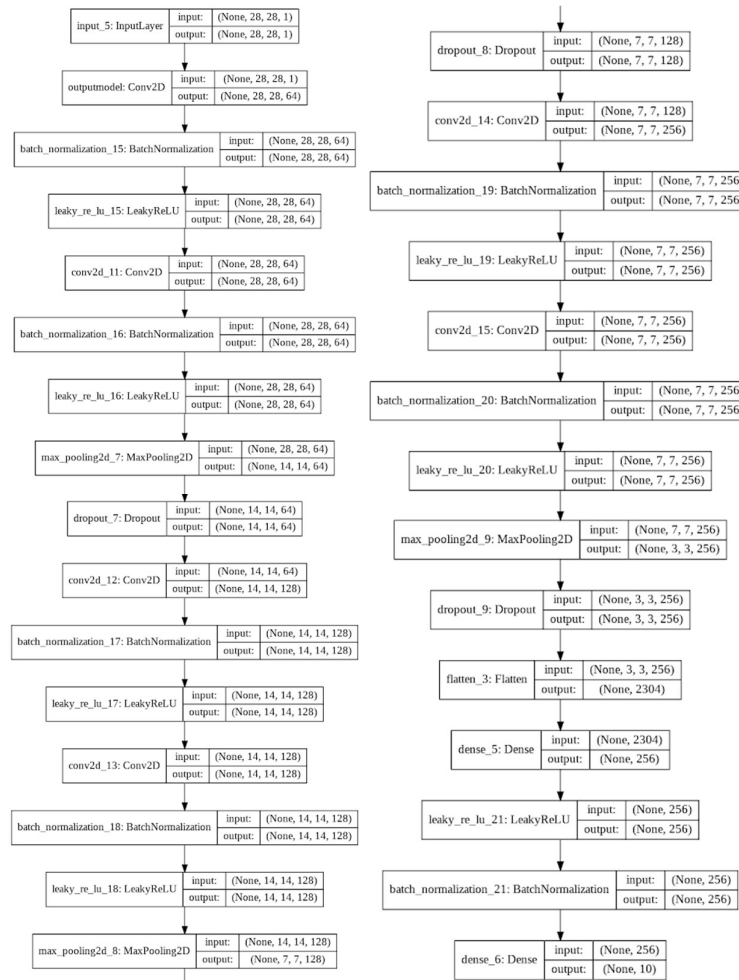


Figure 9.1 The layers of the Convolutional Neural Network used.

9.2 RESULTS AND ANALYSIS OF CNN

Name of model	Test Accuracy	Dig-MNIST accuracy	Parameters	Time per epoch
CNN	98.73	90.46	1,741,514	17s
CapsNet	98.52	91.0	8,384,929	151s

Table 9.1: CNN vs CapsNet results

As we can see from the comparison, the CNN managed to perform better on the test set, with a lower validation and test loss and much a much lesser no of parameters and time taken per epoch.

However, the CapsuleNet had a much better accuracy on the Dig-MNIST database! We speculate that the capsules successfully identified the predicted shapes of the digits and hence could see how their shapes looked, and make accurate predictions on an unseen database.

However, the Convolutional Neural Network seems to be computationally far more efficient, with a fraction of trainable parameters and much less time providing great results on the test set.

This may be because the test set and validation set are a digit test set, which is very well suited for the CNN model.

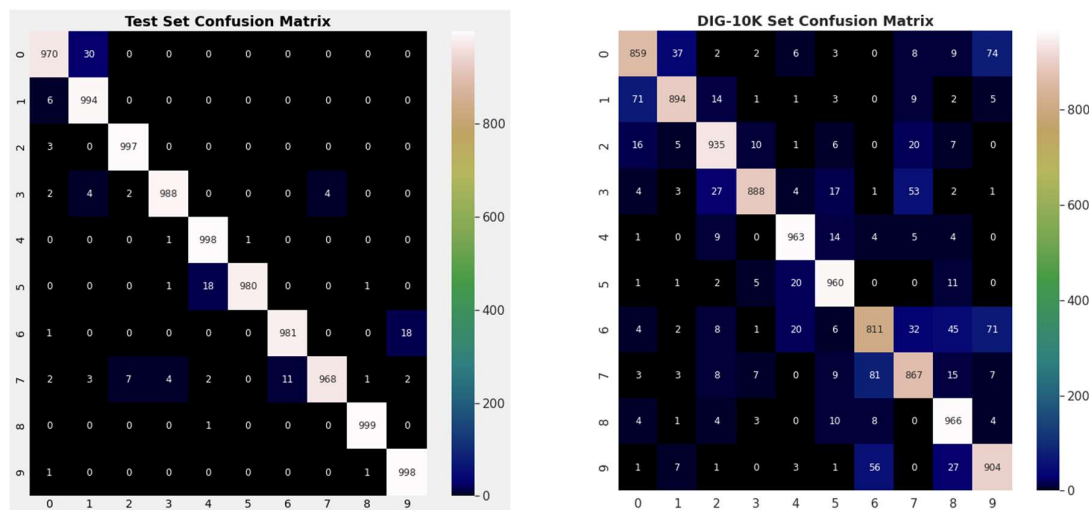


Figure 9.2 Confusion matrices of the CNN model on the test and Dig-10K Sets

Upon comparing these confusion matrices with the ones observed on CapsuleNets, we find that the test set has roughly the same results. However, on the Dig-MNIST database, we find much better results of the CapsuleNet, with less wrong scores and confusion of the digits 6, 7 and 2.

This may be because CNN has lost a lot of information in the pooling layer. Using data augmentation, we manipulated data sets which expanded the scale of our origin data set by downsizing and rotating. Since CNN is less sensitive to spatial orientation and perhaps preserve less accuracy for these changes, it seems that it fared worse in the Dig-MNIST test set.

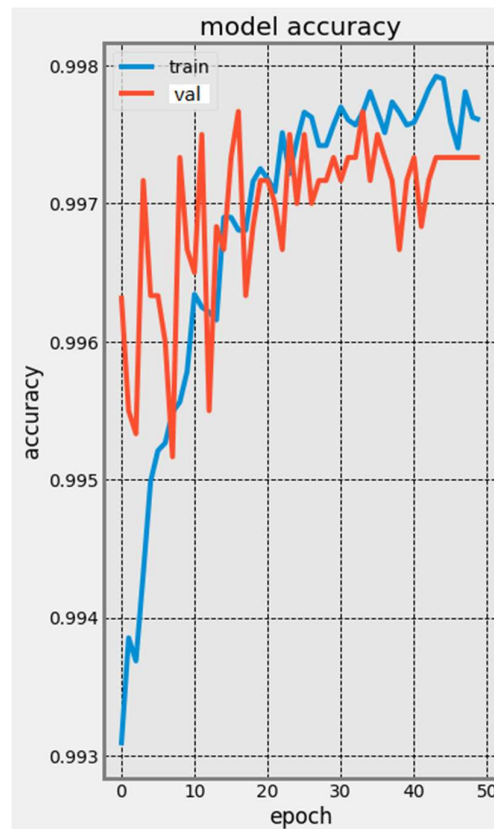


Figure 9.3: Accuracy of the model on the training and the validation sets

The initial accuracy of the testing and validation sets of CNN is greater than the CapsuleNet model, although both remain approximately constant after 35 epochs. The graph of validation set accuracy is comparable for both. The CNN model seems to be better suited to the task as the testing set and validation set instantly receive accurate results. The learning curve of the CapsNet is clearly steeper. Based on these metrics alone, the CNN seems to have an edge.

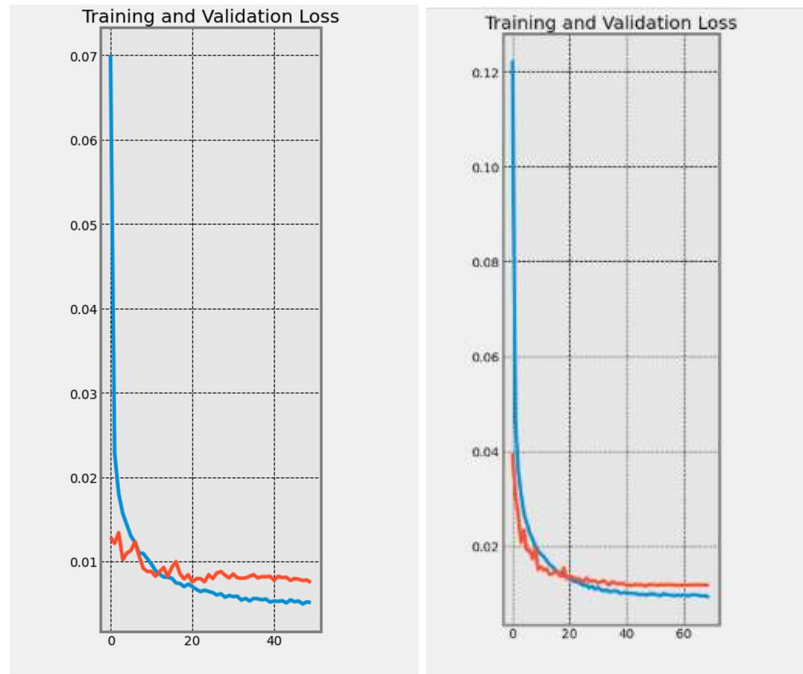


Figure 9.4: Training and validation loss of CNN vs CapsuleNet

From the figure, we observe that while the trajectory of the training loss is around the same, the validation loss of the CapsuleNet is far more stable and smoother than that of the CNN! This is an interesting fact to be observed, and even though the validation accuracy of the CapsNet is less than CNN, the less erratic behavior of the CapsNet loss may be something that needs to be further explored.

CapsNet overcomes the limitations of losing large amounts of data information during the pooling phase of traditional CNNs, as well as being able to handle higher resolution of image classification work. However, the model of CapsNet needs to train far more parameters and takes more time to train (around 10 times than the other two models). A more fair comparison for the CapsNet could maybe be drawn with a database of higher resolution images, which have different features with poses and orientation and more complex shapes. Overall, the MNIST digit databases seem to be better suited for CNN models based on our observations.

10. CONCLUSION

We have been able to do a fairly comprehensive study of the popular literature on Capsule Networks to best understand its need and how it brings a new dimension to solving computer vision problems.

The preservation of the position and pose information leads to promising results for image segmentation and object detection. However, the crowding effect and the high computational costs, along with the slow training of the network may act as a deterrent to widespread adoption of this network.

The comparison of our ConvNet model and our CapsuleNet model shows that CNNs are more suited to the task of classifying digits without requiring much computational power. However, Capsule Nets is an upcoming field of research with promising advancements when orientation of image as a whole matter.

At the same time, both the CapsNet and CNN models created by us give accuracies higher than those mentioned in contemporary research papers.

Our combination of ensemble methods and CapsuleNets provided great accuracy on the Kannada-MNIST, which shows that research on CapsuleNets must be pursued further.

Apart from that, the effectiveness of data augmentation as a tool to increase accuracy became very evident, especially in the case of difficult datasets like Dig-MNIST. Despite that, the inaccurate labelling of data in the dataset served as a major barrier in improving accuracy.

This is still a fairly new topic of research in the field of deep learning and computer vision and we tried to contribute to the best of our abilities to help this architecture gain mainstream popularity. We hope the efficiency of CapsuleNet can be experimented on more datasets with newer and more complex techniques.

11. BIBLIOGRAPHY

- Keras documentation: <https://keras.io/api/>
- Tensorflow documentation: https://www.tensorflow.org/api_docs/python/tf
- <http://sailab.diism.unisi.it/wp-content/uploads/2018/02/Capsule-Networks.pdf>
- <https://www.freecodecamp.org/news/understanding-capsule-networks-ais-alluring-new-architecture-bdb228173ddc/>
- <https://pechyonkin.me/capsules-2/>
- <https://towardsdatascience.com/capsule-neural-networks-part-2-what-is-a-capsule-846d5418929f>
- <https://pechyonkin.me/capsules-3/>
- <https://aboveintelligent.com/ml-cnn-translation-equivariance-and-invariance-da12e8ab7049>
- <https://www.cs.toronto.edu/~hinton/csc2535/notes/lec6b.pdf>
- <https://www.youtube.com/watch?v=pPN8d0E3900>

12. REFERENCES

- [1] Sara Sabour, Nicholas Frosst, Geoffrey E Hinton. "Dynamic Routing between Capsules"
<https://arxiv.org/abs/1904.09546>
- [2] Yann LeCun, Corinna Cortes, Christopher J.C. Burges. "MNIST Handwritten Digit Database"
<http://yann.lecun.com/exdb/mnist/>
- [3] Vinay Uday Prabhu. Kannada-MNIST: "A new handwritten digits dataset for the Kannada language". <https://arxiv.org/abs/1908.01242>]
- [4] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton: "ImageNet Classification with Deep Convolutional Neural Networks". <https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf>
- [5] Haibing Wu, Xiaodong Gu: "Max-Pooling Dropout for Regularization of Convolutional Neural Networks". <https://arxiv.org/ftp/arxiv/papers/1512/1512.01400.pdf>
- [6] Weiwei JIANG, Le ZHANG, Edge-SiamNet and Edge-TripleNet: "New Deep Learning Models for Handwritten Numeral Recognition". <https://doi.org/10.1587/transinf.2019EDL8199>
- [7] Diederik P. Kingma and Jimmy Lei Ba. Adam : "A method for stochastic optimization."
<https://arxiv.org/pdf/1412.6980.pdf>
- [8] Timothy Dozat. "Incorporating Nesterov Momentum into Adam".
<https://openreview.net/pdf?id=OM0jvwB8jlp57ZJjtNEZ>

13. GLOSSARY

- **Softmax:** Function that takes as input a vector of K real numbers and normalizes it into a probability distribution consisting of K probabilities. Usually used as the last layer in neural networks used for classification.
- **CNN:** Convolutional neural network
- **CIFAR 10:** Canadian Institute for Advanced Research dataset, contains 60,000 32x32 color images in 10 different classes of airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks.
- **Max pooling:** Max pooling is a sample-based discretization process to down-sample an input representation reducing its dimensionality and allowing for assumptions to be made about features contained in the sub-regions binned.
- **MNIST Dataset:** Modified National Institute of Standards and Technology database is a large database of handwritten digits that is commonly used for training various image processing systems
- **Affine transformation:** is a linear mapping method that preserves points, straight lines, and planes. Sets of parallel lines remain parallel after an affine transformation. This technique is typically used to correct for geometric distortions or deformations that occur with non-ideal camera angles.
- **Routing Algorithm:** used to decide which route will the data be transmitted to in the next layer.
- **Confusion Matrix:** A confusion matrix is a table that is used to describe the performance of a classification model on a set of test data for which the true values are known.