Capsule Networks: An Alternative Approach to Image Classification Using Convolutional Neural Networks

Karan Sood
Masters in Computer Science
Lakehead University
Student Id:1099550
Section: COMP-5112-WA

***Abstract*—Convolutional neural networks tend to lose lot of information about the image due to their pooling operation, which leads them to misclassify images. In this paper, I discuss how we can preserve the information about the various aspects of the image such as translational information, rotational information etc. by using Capsule networks suggested by the godfather of deep learning- Geoffrey Hinton.**

I. Introduction

A. *What is Deep Learning?*

Deep learning (also known as deep structured learning or differential programming) is part of a broader family of machine learning methods based on artificial neural networks with representation learning. Learning can be supervised, semi-supervised or unsupervised. Deep learning architectures such as deep neural networks, deep belief networks, recurrent neural networks and convolutional neural networks have been applied to fields including computer vision, speech recognition, natural language processing, audio recognition, social network filtering, machine translation, bioinformatics, drug design, medical image analysis, material inspection and board game programs, where they have produced results comparable to and in some cases surpassing human expert performance. Artificial neural networks (ANNs) were inspired by information processing and distributed communication nodes in biological systems. ANNs have various differences from biological brains. Specifically, neural networks tend to be static and symbolic, while the biological brain of most living organisms is dynamic and analog.

Some common application of deep learning are:automatic speech recognition, image recognition, visual art processing, natural language processing, drug discovery and toxicology and recommendation systems to list a few.

Deep learning methods can achieve state-of-the-art results on challenging computer vision problems such as image classification, object detection, and face recognition.We are awash in images: photographs, videos, YouTube, Instagram, and increasingly from live video. Computer Vision, often shortened to CV, is defined as a field of study that seeks to develop techniques to help computers "see" and understand the content of digital images such as photographs and videos.The problem of computer vision appears simple because it is trivially solved by people, even children but teaching computers to do exactly same is quite a challenging task. This is mainly because of following reasons:

- We do not have a strong grasp of how human vision works.
- The complexity inherent in the visual world.
- A true vision system must be able to see in any of an infinite number of scenes and still extract something meaningful.

Deep learning methods are popular, primarily because they are delivering on their promise. Some of the first large demonstrations of the power of deep learning were in computer vision, specifically image recognition. The five promises of deep learning for computer vision are as follows:

- **The Promise of Automatic Feature Extraction:** Features can be automatically learned and extracted from raw image data.
- **The Promise of End-to-End Models:** Single end-to-end models can replace pipelines of specialized models
- **The Promise of Model Reuse:** Learned features and even entire models can be reused across related tasks.
- **The Promise of Superior Performance:** Techniques demonstrate better skill than classical methods on challenging tasks.
- **The Promise of General Method:** A single general method (e.g. convolutional neural networks) can be used on a range of related tasks.

B. *Convolutional Neural Networks*

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics. The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the

human brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

*1) **Why ConvNets over Feed-Forward Neural Nets?:*** In cases of extremely basic binary images, the method of using feed forward neural networks might show an average precision score while performing prediction of classes but would have little to no accuracy when it comes to complex images having pixel dependencies throughout. A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

Consider the following input image: In the figure, we

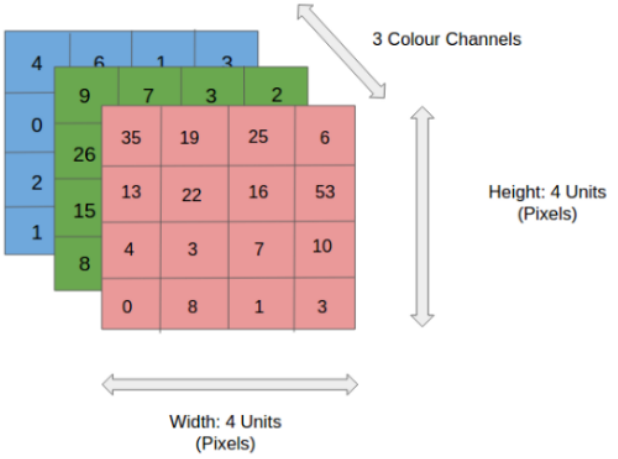

Fig. 1. Input image

have an RGB image which has been separated by its three color planes- Red, Green, and Blue. There are a number of such color spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc. When an image reaches higher dimensions such as 8k (7680 x 4320), then it would turn out to be very computationally expensive if a regular feed-forward network is used. The role of the ConvNet is to reduce the images into a form which is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture which is not only good at learning features but also is scalable to massive datasets.

*2) **The Convolution operation**:* In the figure shown below, the green section resembles our 5x5x1 input image, I. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the Kernel/Filter, K, represented in the color yellow. We have selected K as a 3x3x1 matrix.

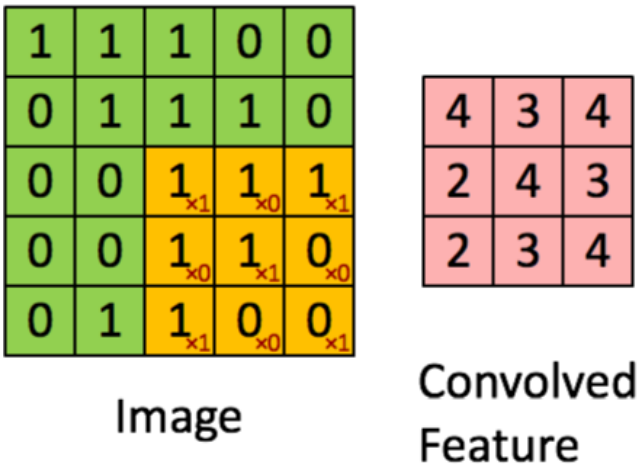

Fig. 2. Convolution operation

The Kernel shifts 9 times because of stride length = 1 (non-strided), every time performing a matrix multiplication operation between K and the portion P of the image over which the kernel is hovering.

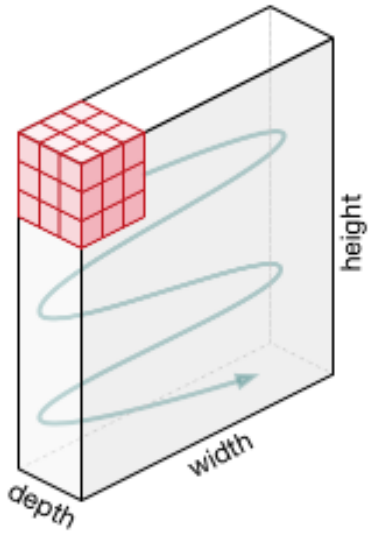

Fig. 3. Kernel slides with stride=1

The filter moves to the right with a certain stride value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same stride value and repeats the process until the entire image is traversed.

C. The Convolution operation

Convolutional networks fail to understand the relationship between simple features in the deeper layers nearer to the input and high level features closer to the output. To solve this, they use max-pooling operation that maps a group of pixels to a particular value, thus, creating high level feature

maps.But in doing so, they tend do lose a lot of information (for example, it has no information about the pose such as translational and rotational information). Thus, mere presence of some features in an image is enough for the CNN to classify it into some category without taking into consideration the relative orientation of those features.
Let us consider a very simple and non-technical example. Imagine a face with components such as the oval face, 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. It does not understand the spatial and relative orientational relationships.

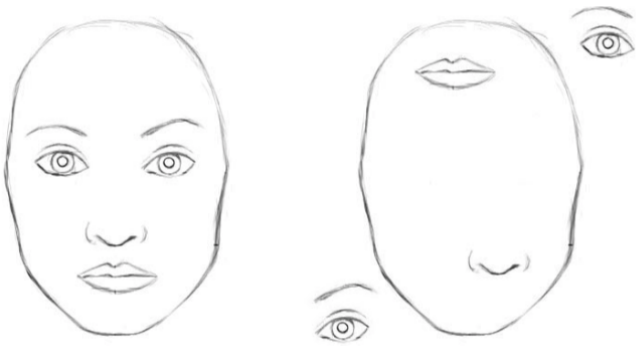

Fig. 4. Non-equivariance in CNN

In this paper, I present my studies on a novel approach developed by Geoffrey Hinton and his colleagues to overcome the drawbacks of CNN discussed above. They called this new type of network architecture- Capsule Networks. In addition to introducing this new network, they also developed an algorithm called Dynamic Routing in order to train these networks.

II. Theory

A. *Inverse Graphics Approach*

Computer graphics deals with constructing a visual image from some internal hierarchical representation of geometric data. Note that the structure of this representation needs to take into account relative positions of objects. That 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 the screen. This is called 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. This is how recognition happens. And the key idea is that representation of objects in the brain does not depend on view angle.

In order to correctly do classification and object recognition, it is important to preserve hierarchical pose relationships between object parts. Capsule networks incorporate these relative relationships between objects and it is represented numerically as a 4D pose matrix.When these relationships are built into internal representation of data, it becomes very easy for a model to understand that the thing that it sees is just another view of something that it has seen before.

Fig. 5. Statue of liberty at different angles

Consider the image above. We can easily recognize that this is the Statue of Liberty, even though both the images show it from different angles. This is because internal representation of the Statue of Liberty in our brain does not depend on the view angle. For a CNN, this task is really hard because it does not have this built-in understanding of 3D space, but for a CapsNet it is much easier because these relationships are explicitly modeled. Another benefit of the capsule approach is that it is capable of learning to achieve state-of-the art performance by only using a fraction of the data that a CNN would use. In this sense, the capsule networks work very much like human brains as human brain requires only few dozens of images to classify the images whereas CNNs require tens of thousands of images to achieve very good performance, which seems to be no better than the brute force approach.

B. *What exactly are Capsule Networks?*

In order to construct an image in a computer, we have to decide some parameters that define various objects present in

the image. For example, consider the image show in below figure.

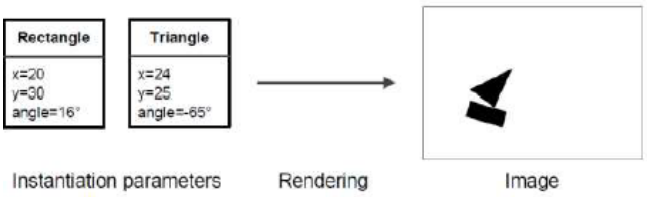

Fig. 6. Parameters instantiation for rendering of images

Now, in order to construct this simple image representing boat, we have to first of all define the parameters of the rectangle and triangle that make up the image. After that, we feed these parameters to a computer software that produces the image as per the defined parameters. This is called rendering. Now consider the reverse scenario as depicted in the image below: Here, we have the image with us and we need to

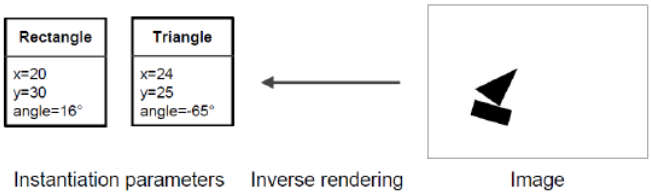

Fig. 7. Estimating parameters given the image

estimate the instantiation parameters that make up various objects present in the image, that is, given the image, we need to find the x and y co-ordinates as well as the rotational angle of the boat and the triangle that make up the image. This process is called Inverse graphics approach which Hinton had talked about. Capsule networks are networks that do inverse rendering and try to identify the relationship between various objects that make up the image.

C. *How are capsules represented?*

The capsules in the capsule networks are represented by something called activation vectors. Just like any ordinary vector, these activation vectors also have two fundamental characteristics- the magnitude and the direction, as shown in the **fig 8**

The length of the activation vector represents the probability of an object being present at a particular location in the image- zero length indicating absence whereas length of one indicating absolute confidence about the presence of the object. The direction on the other hand tells us about the pose of the image. It gives us information about the rotational and translational aspects among many others.

Since we know that measure of probability can never be greater than one, therefore, we have to make sure that the

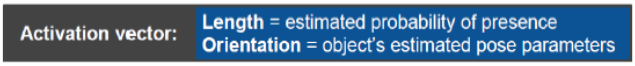

Fig. 8. Activation vector

length of each activation vector produced is between 0 and 1. To achieve this, we use a function called squash function which is described by the below formula:

$$\text{squash}(\mathbf{s}) = \frac{\|\mathbf{s}\|^2}{1+\|\mathbf{s}\|^2}\frac{\mathbf{s}}{\|\mathbf{s}\|+\epsilon}$$

Fig. 9. Squashing the activation vector

In order to handle the case where the object might have zero probability to be present at a particular location, we need to add a non-zero parameter called 'epsilon' to denominator so that even if we have zero chance of observing an object we do not end up with a case where we have zero by zero division.

D. *Concept of equivariance*

The main drawback of convolutional neural networks is that it tends to lose lot of information due to pooling operation and hence, it is not able to preserve pose information of an object in the image. This is known as non-equivariance. Capsule networks tend to preserve the pose information.

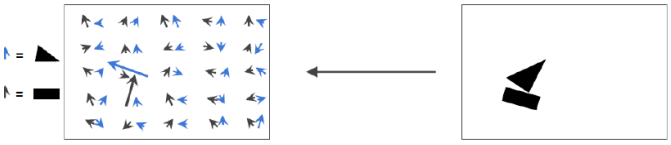

Fig. 10. One possible orientation

If we look at **Fig 10**, then we can see that to represent the boat in the image, we have 50 corresponding capsules

on the left. Among these capsules, there are two capsules or activation vectors that have larger length as compared to other capsules in the same figure. This represents higher probability of finding the triangle and rectangle at those specific positions.

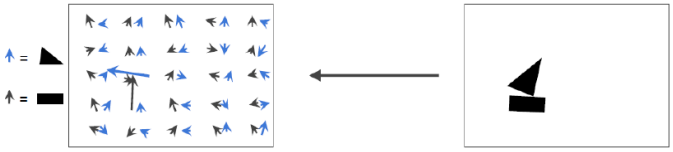

Fig. 11. Changed orientation

If we look at **Fig 11**, then we can observe that even on slightly changing the orientation of the boat, there is a corresponding change in the orientation of these two activation vectors as well, which demonstrates the fact that these capsules are able to capture the pose information with the help of the activation vectors. Thus, we can say that capsules preserve the pose information and are able to overcome the drawbacks of traditional convolutional neural networks.

E. How do capsules work?

If we consider the working of an artificial neuron, then we can easily observe that there are three basic steps that govern the working of an artificial neuron. These are as follows:

1) **Scalar product of weights:** In this step, each artificial neuron receives inputs from all other neurons it is connected to in the form of scalar values and multiplies them with the corresponding weights to obtain scalar products..
2) **Sum of products:** In this step, all the products that have been calculated in the above step are added together to obtain the weighted sum of the signals that the neuron has received.
3) **Non-linear transformation:** In this last step, the weighted sum obtained in the previous step is passed through one of the many activation functions available. This step is required to add some non-linearity to the input so that neural network is able to capture non-linear trends in the data as well. If we do not use any activation function, then a neural network would only be able to identify the linear trends in the data and would lead to problems such as heteroskedasticity, that is, non-constant variance in the error terms leading to unstable confidence intervals.

A capsule has these same transformations in the vector form and an additional step where all the input vectors received from the lower level neurons are multiplied together. These steps are discussed in detail below.

*1) **Matrix multiplication of the input vectors**:* Consider three inputs from lower level capsules entering the capsule 'j' present in the layer above. The outputs from the lower level capsules are represented as vectors u1, u2 and u3. Lengths of these vectors encode probabilities that lower-level capsules detected their corresponding objects and directions of the vectors encode some internal state of the detected objects. Let us assume that lower level capsules detect eyes, nose and mouth respectively and output capsule detects face.

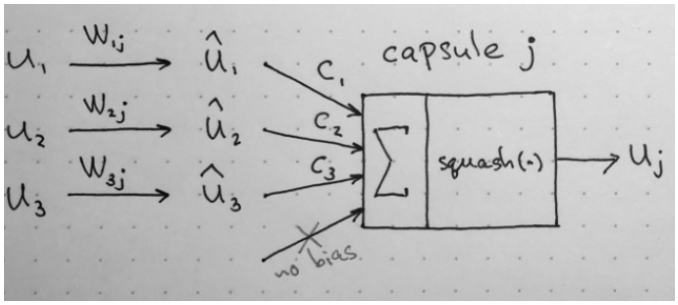

Fig. 12. Working of capsule

These vectors then are multiplied by corresponding weight matrices **'W'** that encode important spatial and other relationships between lower level features (eyes, mouth and nose) and higher level feature (face). For example, matrix **W2j** may encode relationship between nose and face: face is centered around its nose, its size is 10 times the size of the nose and its orientation in space corresponds to orientation of the nose, because they all lie on the same plane. Similar intuitions can be drawn for matrices **W1j** and **W3j**. After multiplication by these matrices, what we get is the predicted position of the higher level feature. In other words, **u1hat** represents where the face should be according to the detected position of the eyes, **u2hat** represents where the face should be according to the detected position of the nose and **u3hat** represents where the face should be according to the detected position of the mouth.
If the predictions made by these three capsules point at the same position and state of the face, then the network concludes that it must be a face there.

*2) **Scalar weighting of input vectors**:* In artificial neural networks, the neuron receiving the input signal from the neurons present in the previous layer weighs them according to weights of the corresponding neuron that connects them. These weights are learned by the network during back-propagation. But in case of capsule networks these weights are decided by a process called Dynamic Routing. The higher level capsules receive input from many lower level capsules. When the lower level capsules agree on a particular prediction, they tend to form a cluster in the higher level capsule. When the output of the lower level capsule is multiplied by the weight matrix learned by the network, it tends to land in the cluster space of the higher level capsule. The lower level capsule output would be routed to that particular capsule where it tends to agree with predictions made by the other low-level capsules.
In order to understand this better, we can consider the figure given below where we have two higher level capsules and one lower level capsule whose output we have to decide the routing for.

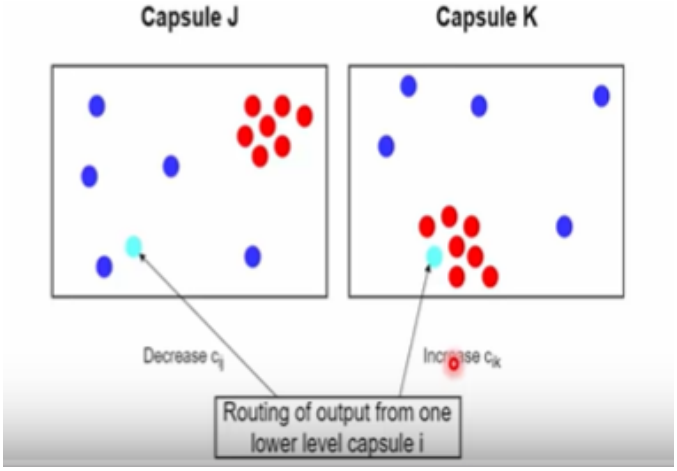

Fig. 13. Routing of input signals to capsules

In the image above, we have one lower level capsule that needs to "decide" to which higher level capsule it will send its output. It will make its decision by adjusting the weights **C** that will multiply this capsule's output before sending it to either left or right higher-level capsules **J** and **K**. Now, the higher level capsules already received many input vectors from other lower-level capsules. All these inputs are represented by red and blue points. Where these points cluster together, this means that predictions of lower level capsules are close to each other. This is why, for the sake of example, there is a cluster of red points in both capsules J and K. When we multiply the lower level capsule's output with the weight matrix, it lands far from the cluster of red dots in capsule **J** whereas for other capsule, that is capsule **K**, it lands near the cluster of red dots. So, the output of the lower level capsule would be routed to capsule **K** rather than capsule **J**.

*3) **Sum of the weighted input vectors**:* This is same as the regular artificial neurons where we simply sum up the weighted outputs. The only difference in the capsule networks is that we have the addition of vectors here.

*4) **Squashing of the vectors**:* Just like I mentioned that the length of the activation vector represents the probability of finding an object at a particular position in the image, it needs to be ensured that it does not exceed one as the probability can never be greater than one. For this purpose, we make use of squash function that takes in a vector as an input and produces another vector that has length between 0 and 1 and without any change in the original direction. This operation is known as the squashing operation.

F. The dynamic routing between the capsules

*1) **The calculation of actual output of target capsules**:* Consider the following picture showing a simple boat made up of a rectangle and a triangle.

In this picture, we have to route the predicted output from the lower level capsules to the higher layer which contains two capsules because there are two possibilities for the object-one is the object being a house and the other one is the object being a boat. In order to decide the path, the following steps are followed:

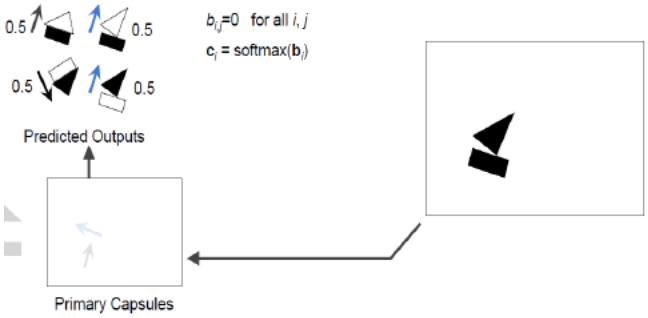

Fig. 14. Routing mechanism

- **Initializing the raw routing weights to zero:** In this step, all the weights are initialized to zero. These are known as the raw routing weights represented as $\mathbf{b}_j$.
- **Calculating the actual routing weights:** The second step is to apply soft-max activation function to the raw routing weights calculated in the previous step to obtain the actual routing weights that are represented as $\mathbf{c}_j$.
- **Weighted sum calculation:** In this step, the routing weights obtained in the previous step are multiplied by the predicted output. In doing so, it might happen that the length of the vectors become greater than one, therefore, we need to apply squash function one more time to satisfy the probability constraint. This gives us the calculated actual output of the higher level capsules which is represented as $\mathbf{v}_j$.

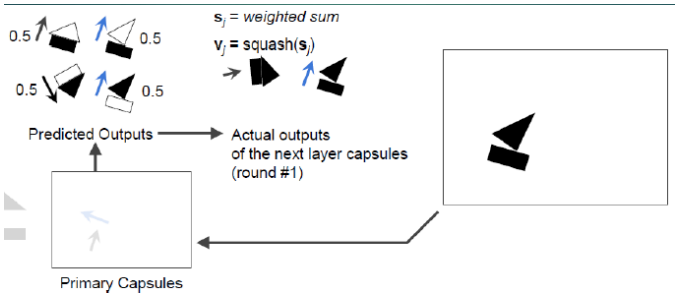

Fig. 15. Actual output calculation

The figure above shows how the actual output is calculated using the routing mechanism.

After calculating the actual output, we need to update the routing weights to make our model learn. In order to update the routing weights, we first of all calculate the degree of similarity between the predicted output and the actual output. This is done by calculating the dot product of the predicted vector and the calculated vector. When there is agreement, that is angle between the two vectors is less than 90 degrees, its cosine value is positive and there is a corresponding increase in the routing weight. On the other hand, when the angle between the

two vectors is greater than 90 degrees, then cosine of that angle becomes negative resulting in decrease in the corresponding routing weights. In below figure, there is similarity between the predicted and the actual output and hence, there is an increase in the routing weights.

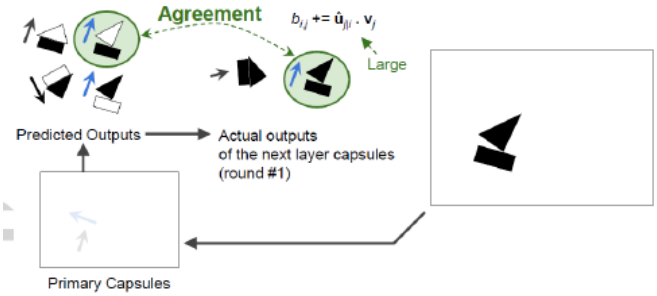

Fig. 16. Routing weights increase

The dissimilarity between the predicted output and the actual output causes the routing weights to decrease as shown in the figure below.

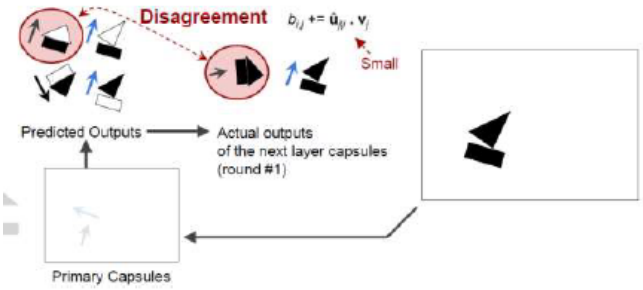

Fig. 17. Routing weights decrease

*2) **Convergence of the algorithm**:* This whole process described above is just a single iteration in dynamic routing mechanism. It repeats itself over a couple of times till there is no change in the routing weights for specified number of iterations or else the maximum number of iterations has been reached, whichever happens earlier.
The dynamic routing, thus, helps us in preventing noise in the system by properly routing the outputs of lower level capsules to appropriate capsules in the higher layer.

III. Methodology

I have used TensorFlow to implement the capsule network.

A. *Feeding the input images*

My first step was to create tensor flow placeholder that would directly feed the input images of dimensions (28 x 28 x 1) to the network. Since I am working with gray-scale images, therefore, the last channel has the value 1. If there were coloured images, then, the last channel value would have been 3 corresponding to the red, green and blue channels.

B. *Building the primary layer*

The second step was to build the primary capsule layer that would try to identify different types of information about the objects present in the scene. For this purpose, I used two convolutional layers. To the first convolutional layer, I directly fed the input images. The output of this convolutional layer was fed as an input to the second convolutional layer that was configured to output 256 feature maps with each of them containing 36 scalars. The feature maps were reshaped so that I now had 32 maps containing 6 by 6 grid of 8 dimensional vectors. This is illustrated in the image below.

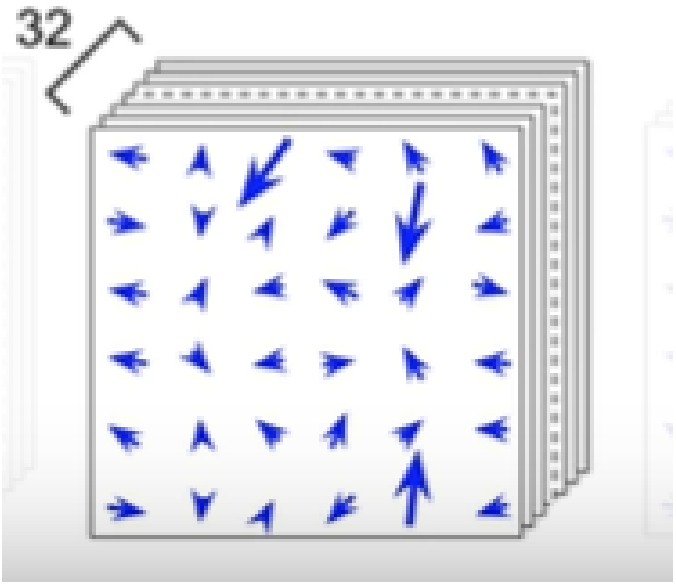

Fig. 18. Convolutional feature maps reshaped

C. *Modifying the squash function*

The squash function that I defined in **Fig 9** has a small problem when I tried implementing it in the TensorFlow. The problem is that if any of the vectors in the equation represented by **Fig 9** become zero, that is when we have zero chance of observing an object in the given scene, then the gradients can not be computed and the 'norm' function of TensorFlow gives us 'NaN' values due to which the network literally becomes dead. In order to solve this problem, the squash function can be modified as shown in **Fig 19**.

$$\|\mathbf{s}\| \approx \sqrt{\sum_i s_i^2 + \epsilon}$$

Fig. 19. Modified squash function to handle **NaN** values

D. *Implementing the digit capsules*

Since there were ten classes to be identified (numbers **0** to **9**), therefore, there were 10 capsules in the output layer. Each of these capsules were 16-dimensional vectors that contained full pose information (such as the skewness, rotation relative to the axis etc.) of the objects present in the image to be classified. Now, I needed to compute the output of the digit capsules. This can be done in two steps.

*1) **Making predictions for the digit capsules**:* The first step was to predict the output for each of the capsules present in the digit capsule layer. The prediction is made by computing the product of transformation matrix learned during the training process and the output of the primary level capsules. The transformation matrix learns the part-whole relationship between the objects present in the image and the image itself. This process is repeated for every pair possible between the lower level capsules and the number of capsules present in the higher layer. For example, here in the lower layer, there are (32 x 36 = 1152) capsules present. Therefore, there would be these many predictions for every digit capsule in the higher layer. Since there are 10 digit capsules in total, therefore, we have total of 11520 predictions. This is shown in the figure below.

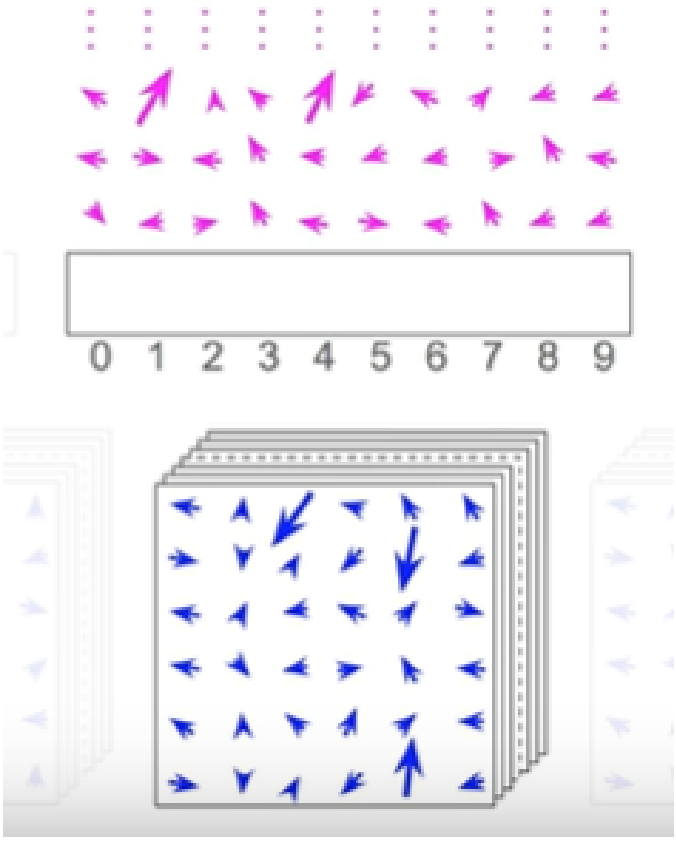

Fig. 20. Predicted output for the higher-level capsules

*2) **Dynamic routing of the outputs of lower level capsules**:* After calculating the predicted outputs for the higher-level capsules in the first step, we need to make sure that they reach appropriate capsules in the higher layer so that the noise in the system can be prevented. This is done by an algorithm called routing by agreement which I have discussed in the theory part. Visually, it looks something like this.

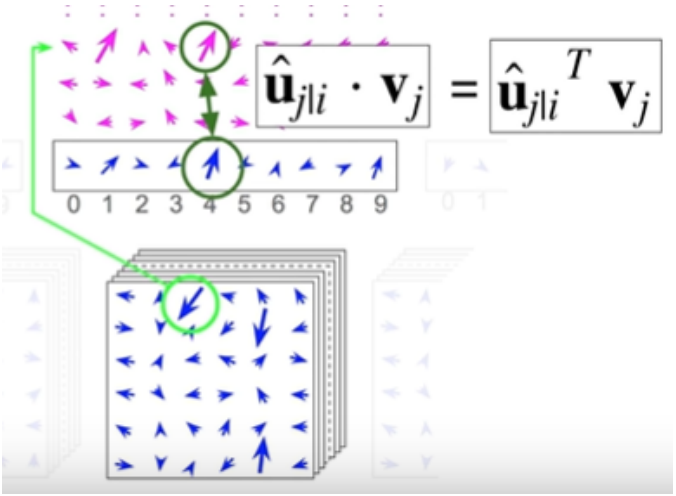

Fig. 21. Routing of the predicted outputs

E. *Estimating the class probabilities*

After calculating the outputs of the digit capsules, I calculated the length of the those output vectors that would give me the probabilities of an instance to belong to each of the 10 classes, that is, the probability of an instance to be one of the digits. After computing these probabilities, the class of a given instance is determined by finding the digit capsule that outputs the maximum value among all the capsules. For example- if the length of the output vector corresponding to the digit capsule 7 is found to be maximum, then we can say that the image is of digit 7. One important point that should be taken care of is that we are not using a soft-max activation in the last layer due to which the probabilities may not add up to one, But this gives us an additional advantage of detecting multiple images in the scene corresponding to the digit capsules that output the maximal probability values. The next step was to improve the model using some loss function. For this I used something suggested in the paper that first introduced the concept of capsule networks and dynamic routing algorithm. The loss is called margin loss and is given by the following equation:

$$L_k = T_k \max(0, m^+ - \|\mathbf{v}_k\|)^2 - \lambda(1 - T_k)\max(0, \|\mathbf{v}_k\| - m^-)^2$$

Fig. 22. The Margin loss for training the model

The term **T** represents the probability of an image to belong to class **'k'.**. For a given image instance, it is equal to 1 if the image of class **'k'** is present in the image, else it is equal to 0. In order to minimize this margin loss, there is a constraint which says that if an instance belongs to class **'k'**, then the value of **'v'** for class **'k'** should be greater than **0.9**, else it should be less than **0.1**

The whole process up till now can be visualized as follows:

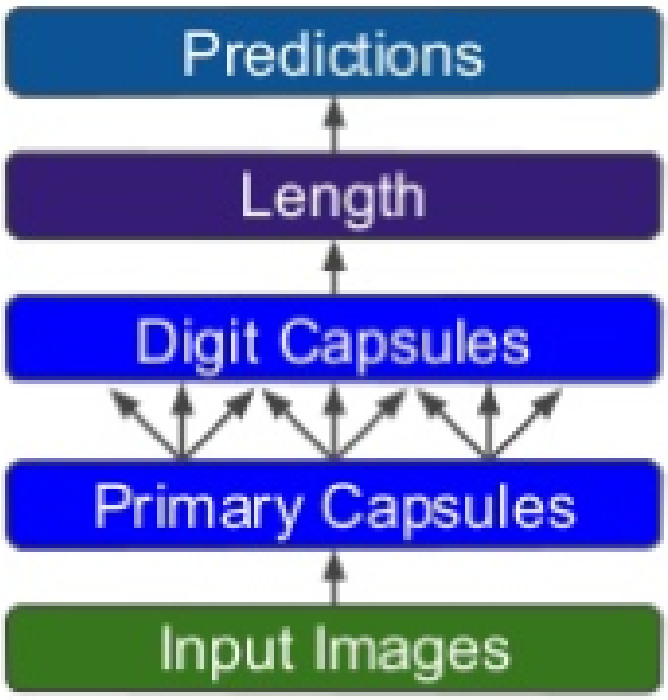

Fig. 23. A snapshot of the process

F. *Computing reconstruction loss*

To compute the reconstruction loss, I had to first construct the image from the information that the network has learnt. I used a decoder for this purpose. The decoder has three fully connected layers also called dense layers with first layer having 512 hidden neurons, the second layer having 1024 hidden units and the final one having 784 hidden units. The decoder produced a 784-dimensional vector containing the pixel intensity values for each 28 by 28 image instance that was fed to the capsule network. The reconstruction loss was computed by squaring the difference between the original image and reconstructed image. Since the original image had dimensions (28x28x1), therefore, it had to be first converted into 784-dimensional vector. The total loss was obtained by adding together the margin loss and the reconstruction loss but it was scaled down in a manner to let the margin loss dominate the training process of the model. I used **"Adam"** optimizer with default settings to optimize the training process. One important thing suggested in the paper was that instead of sending all the outputs of the capsule network to the decoder network, we must send only the output vector of the capsule that corresponds to the target digit. This was done by setting zeros for all the values of the digit capsules except for the one that actually represented the image. For testing instances, since labels are not known beforehand, the predictions for a particular class was used to decide the mask.
The entire process of training from the instance an input image is fed to the network till the model fits to the data can be visualized with the help of the following figure:

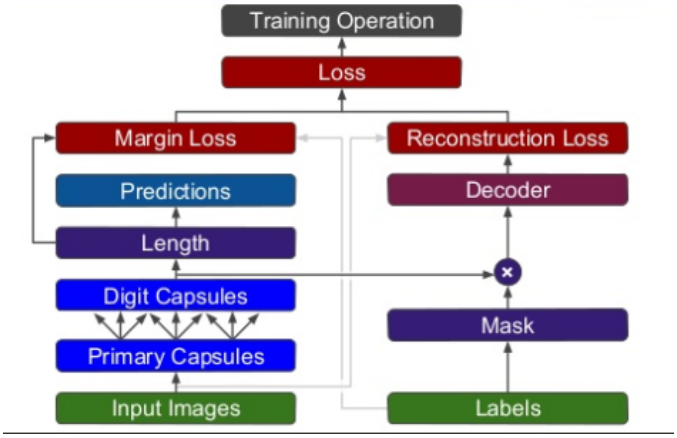

Fig. 24. The training process

The above image demonstrates the predictions made by the model when it is dealing with the training instances. Here, the class labels are used to create the mask. When the model has to deal with the testing instances where the labels are not known, the mask is created by using the predictions of the model. This is illustrated in the figure below.

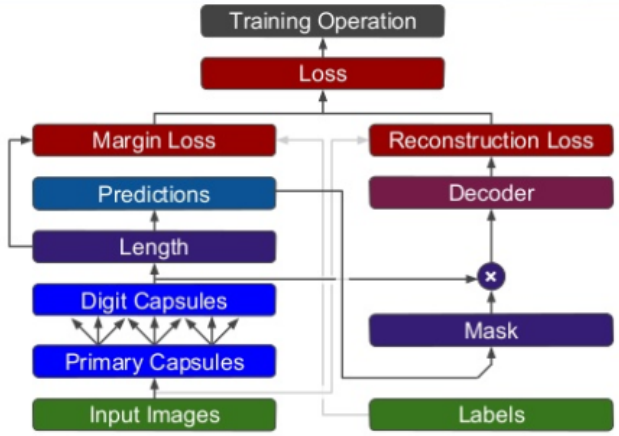

Fig. 25. The entire testing process

IV. Results

A. *Training accuracy*

I used **MNIST** dataset for training the model. It contained 60000 training images and 10000 test images. I did not do hyper-parameter tuning or used any dropout layer. I just trained the model for a few epochs, each time measuring the accuracy on the validation set and saving the model if the validation loss happened to be lowest so far. This is method is generally called **'Early stopping'**. The validation set contained 5000 images and the model computed the mean accuracy and mean loss value at the end of each epoch. Since TensorFlow has been used to implement the code, GPUs or even better TPUs (tensor

processing units) can greatly enhance the speed of training and the corresponding time would be greatly reduced. After a few epochs, there was considerable learning by the model and it was giving an accuracy of **99** percent on the validation set containing 5000 images. This is shown in the below figure:

```
INFO:tensorflow:Restoring parameters from ../input/CapsNets_implementation
Epoch: 1  Val accuracy: 99.4400%  Loss: 0.007998
Epoch: 2  Val accuracy: 99.3400%  Loss: 0.007959
Epoch: 3  Val accuracy: 99.4000%  Loss: 0.007436
Epoch: 4  Val accuracy: 99.4000%  Loss: 0.007568
Epoch: 5  Val accuracy: 99.2600%  Loss: 0.007464
Epoch: 6  Val accuracy: 99.4800%  Loss: 0.006631
Epoch: 7  Val accuracy: 99.4000%  Loss: 0.006915
Epoch: 8  Val accuracy: 99.4200%  Loss: 0.006735
Epoch: 9  Val accuracy: 99.2200%  Loss: 0.007709
Epoch: 10  Val accuracy: 99.4000%  Loss: 0.007083
```

Fig. 26. Training epochs

B. Testing accuracy

After training the model for a few epochs, I evaluated the model on the test set that contained 10000 images as is provided by the creators of **MNIST** dataset. The test accuracy of the model was found to be **99.5** percent. Earlier, I found it a little strange having obtained better accuracy on the test set than the training set. But, I researched and I found that sometimes model is indeed able to better chase the errors in the test set depending on how we split the data. The accuracy on the test set is shown in the figure below.

```
INFO:tensorflow:Restoring parameters from ../input/CapsNets_implementation
Final test accuracy: 99.5300%  Loss: 0.006631
```

Fig. 27. Accuracy on the test set

C. Predictions

After training my model, I decided to make some predictions to have a better idea about how my model was performing. The model's ability in making predictions on new data is shown in **Fig 28** and **Fig 29** on the right hand side.

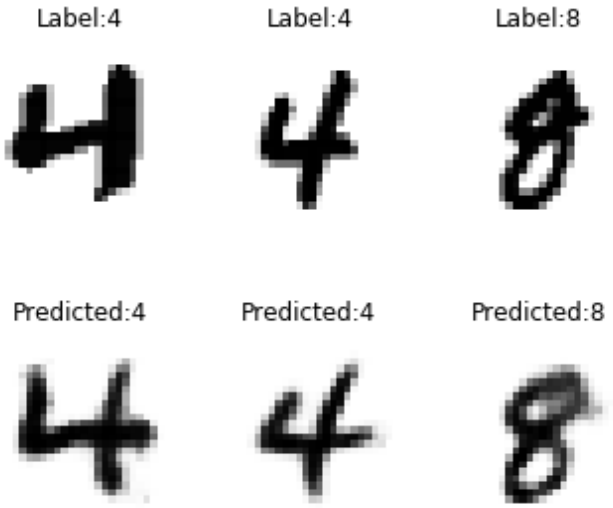

Fig. 28. Predictions-1

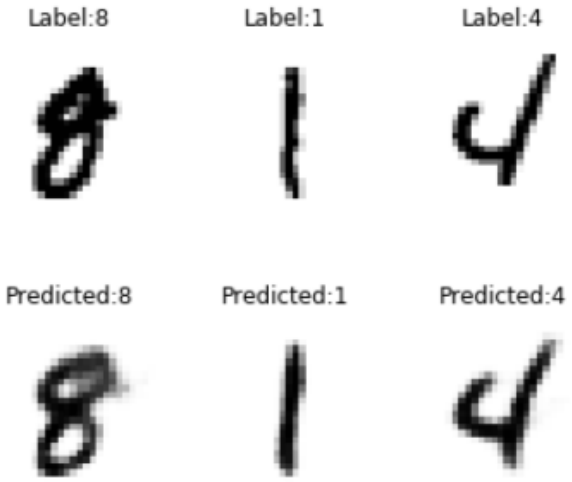

Fig. 29. Predictions-2

V. SUMMARY

The traditional use of convolutional neural networks can be useful to some extent when we want to just label the images and do not care much about the features that make up the image. But, when it comes to tasks such as object detection or image segmentation, then convolutional nets tend to fail miserably due to loss of information during the pooling operation that reduces the dimensionality of the feature maps created during the convolution operation. Thus, we can say that convolutional networks are **non-equivariant** in nature. This drawback can be overcome by the use of capsule networks suggested by Geoffrey Hinton. The capsule networks make use of capsules which are nothing but the activation vectors whose length represent the probability of presence of an object in the image and the direction represent the orientation. Depending upon the number of dimensions used to represent the activation vector, we can capture lot of pose information such as translational information, rotational information, skewness, thickness of the image etc. Whenever there is even a minute change in the image, there is a corresponding change in the values of the activation vector as well. This way the pose information of the image is preserved and we say that capsule networks are **equivariant** in nature.

VI. FUTURE WORK

There is lot of scope for improvement in the design of the capsule networks. Although the capsule networks have been able to achieve remarkable performance on the **MNIST** dataset, but the performance is relatively average as far as the other datasets such as **CIFAR10** or **CIFAR100** are concerned. We also do not have any idea about the performance of capsule networks on large image datasets such as **ImageNet**. Another area that needs improvement is the dynamic routing of the capsule networks as they significantly increase the training

time of the network, One last thing that I would like to mention is that capsule networks are not able to detect identical objects close to each other. But, this may or may not be a problem as even human eyes are not very good at it.

REFERENCES

[1] Geoffrey E.Hinton, Google Brain, Toronto, Sara Sabour, Nicholas Frost, Dynamic Routing Between Capsules, NIPS 2017.

[2] G. E. Hinton, A. Krizhevsky S. D. Wang, Transforming Auto-encoders, 2011.

[3] Wei Zhao, Min Yang, Shenzen Institutes of Advanced Technology, Chinese Academy of Sciences, Jiambo Ye, Pennsylvania State University, Zeyang Lei, Graduate School at Shenzhen, Tsinghua University, Soufei Zhang, Nanjing University of Posts and Telecommunications, Zhou Zhao, Zhejiang University, Investigating Capsule Networks with Dynamic Routing for Text Classification, 2018.

[4] Mercedes E. Paoletti, Student Member, IEEE, Juan M. Haut, Student Member, IEEE, Ruben, Fernandez-Beltran, Javier Plaza, Senior Member, IEEE, Antonio Plaza, Fellow, IEEE, and Filiberto Pla.

[5] Ningyu Zhang, Artificial Intelligence Research Institute, Zhejiang Lab, China, Shumin Deng, College of Computer Science and Technology, Zhejiang University, China, Alibaba-Zhejiang University Frontier Technology Research Center, China, Zhanlin Sun, College of Computer Science and Technology, Zhejiang University, China, Xi Chen, College of Computer Science and Technology, Zhejiang University, China, Wei Zhang, Alibaba-Zhejiang University Frontier Technology Research Center, China, Alibaba Group, China, Huajun Chen, College of Computer Science and Technology, Zhejiang University, China.

[6] Steve Lawrence, Member, IEEE, C. Lee Giles, Senior Member, IEEE, Ah Chung Tsoi, Senior Member, IEEE, Andrew D. Back, Member, IEEE, Face Recognition: A Convolutional Neural-Network Approach, IEEE TRANSACTIONS ON NEURAL NETWORKS, VOL. 8, NO. 1, JANUARY 1997.

[7] Patrice Y. Simard, Dave Steinkraus, John C. Platt, Microsoft Research, One Microsoft Way, Redmond WA 98052, Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis.

[8] Alex Krizhevsky, University of Toronto, Ilya Sutskever, University of Toronto, Geoffrey E. Hinton, University of Toronto, ImageNet Classification with Deep Convolutional Neural Networks.

[9] Tianjun Xiao, Institute of Computer Science and Technology, Peking University, Yichong Xu, 2Microsoft Research, Beijing, Kuiyuan Yang, Microsoft Research, Beijing, Jiaxing Zhang, Microsoft Research, Beijing, Yuxin Peng, Institute of Computer Science and Technology, Peking University, Zheng Zhang, Institute of Computer Science and Technology, Peking University, The Application of Two-level Attention Models in Deep Convolutional Neural Network for Fine-grained Image Classification.

[10] Dumitru Erhan, Christian Szegedy, Alexander Toshev, and Dragomir Anguelov, Google, Inc, 1600 Amphitheatre Parkway, Mountain View (CA), 94043, USA.

[11] Wanli Ouyang, Ping Luo, Xingyu Zeng, Shi Qiu, Yonglong Tian, Bhuvana Ramabhadran1Hongsheng Li, Shuo Yang, Zhe Wang, Yuanjun Xiong, Chen Qian, Zhenyao Zhu, Ruohui Wang, Chen-Change Loy, Xiaogang Wang, Xiaoou Tang, the Chinese University of Hong Kong, DeepID-Net: multi-stage and deformable deep convolutional neural networks for object detection.

[12] Jifeng Dai, Microsoft Research, Yi Li, Tsinghua University, Kaiming He, Microsoft Research, Jian Sun, Microsoft Research, R-FCN: Object Detection via Region-based Fully Convolutional Networks.

[13] K. Mikolajczyk, B. Leibe, and B. Schiele. Multiple object class detection with a generative model. In CVPR, volume 1, pages 26–36. IEEE, 2006.

[14] Tara N. Sainath, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A, Abdel-rahman Mohamed, Department of Computer Science, University of Toronto, Canada, Brian Kingsbury, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A, Bhuvana Ramabhadran1, IBM T. J. Watson Research Center, Yorktown Heights, NY 10598, U.S.A, DEEP CONVOLUTIONAL NEURAL NETWORKS FOR LVCSR.

[15] Amara Dinesh Kumar, R.Karthika , Latha Parameswaran , Department of Electronics and Communication Engineering , Amrita School of Engineering, Coimbatore , Amrita Vishwa Vidyapeetham,India.

[16] Johannes Stallkamp, Marc Schlipsing, Jan Salmen, and Christian Igel. The german traffic sign recognition benchmark: a multi-class classification competition. In Neural Networks (IJCNN), The 2011 International Joint Conference on, pages 1453–1460. IEEE, 2011.