Comparative Study between CNN and Capsule Network

PC 403 B.Tech Mini Project

Keya Desai (201501012)
Prakruti Joshi (201501044)

Guide: Prof. MV Joshi

1. Abstract

We have done a comprehensive study of Capsule Network and its Architecture. Capsnet proposes a solution for the widely prevailing CNN's drawback of max pooling. Capsnet preserves the spatial relationship between the features overcoming this problem of max pooling. We have thus done a comparative study between CNN and Capsule Network on multiple datasets. We trained CNN and CapsNet on MNIST and Cifar10 dataset and presented the results. To extend the scope of Capsule Networks, its pros and cons have been weighed. A literature survey is conducted on computer vision problems such as image segmentation and object detection and using that, we have proposed a future scope of Capsnet in Object Detection.

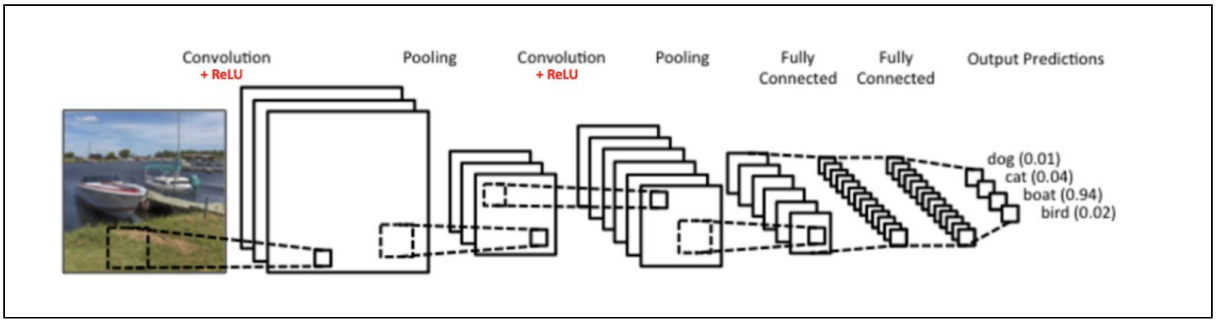
2. Introduction

CNN is good at detecting features but less effective at exploring the spatial relationships among features (perspective, size, orientation).Internal data representation of a convolutional neural network does not take into account important spatial hierarchies between simple and complex objects. In Capsnet these relationships are explicitly modeled.

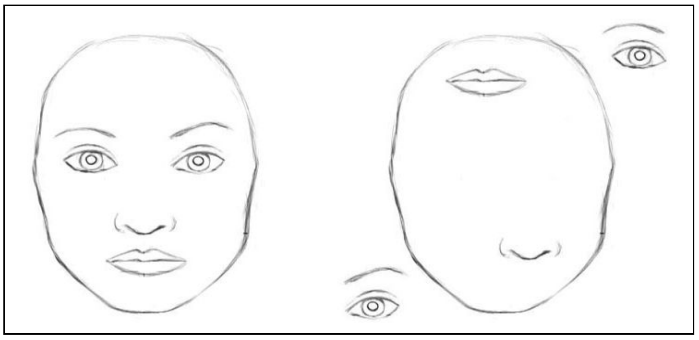
Max pooling

The max pooling in a CNN handles translational variance. Even a feature is slightly moved, if it is still within the pooling window, it can still be detected. Nevertheless, this approach keeps only the max feature (the most dominating) and throws away the others. Capsules maintain a weighted sum of features from the previous layer. Hence, it is more suitable in detecting overlapping features. For example detecting multiple overlapping digits in the handwriting.

CNN Architecture



CNN will detect both as faces.



3. Capsule Network

Capsule

A capsule network is composed of number of capsules rather than neurons. A capsule is a small group of neurons that learns to detect a particular object (e.g., a rectangle) within a given region of the image, and it outputs a vector (e.g., an 8-dimensional vector). Length of the activation vector represents the estimated probability that the object is present, and orientation (e.g., in 8D space) encodes the object's pose parameters (e.g., precise position, rotation, etc.). So, if the length is small, it implies that the capsule did not detect anything whereas a vector with greater length implies that the capsule found what they were looking for (e.g a rectangle). The instantiation parameters maybe orientation of the object, it's thickness, how skewed it is, it's rotations or translations.

Layers

Much like a regular neural network, a CapsNet is organized in multiple layers. The capsules in the lowest layer are called primary capsules: each of them receives a small region of the image as input (called its receptive field), and it tries to detect the presence and pose of a particular pattern, for example a rectangle. Capsules in higher layers,

called routing capsules, detect larger and more complex objects, such as boats and houses.

The primary capsule layer is implemented using a few regular convolutional layers. The convolution layers gives a array of feature maps. This array can be reshaped to get a set of vectors for each location. For example, in the paper, they use two convolutional layers that output 256 6x6 features maps containing scalars. They reshape this output to get 32 6x6 maps containing 8-dimensional vectors. Finally, they use a novel squashing function to ensure these vectors have a length between 0 and 1 (to represent a probability). And that's it: this gives the output of the primary capsules.

The capsules in the next layers also try to detect objects and their pose, but they work very differently, using an algorithm called **routing by agreement.**

Dynamic routing

A transformation matrix Wij to the capsule output ui of the previous layer. Prediction vector û j|i is computed as:

$$\hat{u_{j|i}} = W_{ij} u_i$$

The activity vector vj (the capsule output) is:

$$s_j = \sum_i c_{ij} \hat{u_{j|i}}$$

$$v_j = \frac{\|s_j\|^2}{1 + \|s_j\|^2} \frac{s_j}{\|s_j\|}$$

Sj is the total input to higher layer capsule.
Intuitively, prediction vector $\hat{u}_{j|i}$ is the prediction (**vote**) from the capsule ii on the output of the capsule jjabove. If the activity vector has close similarity with the prediction vector, we conclude that both capsules are highly related. Such similarity is measured using the scalar product of the prediction and the activity vector.

$$b_{ij} \leftarrow \hat{u_{j|i}} \cdot v_j$$

Therefore, the similarity score bij takes into account on both likeliness and the feature properties, instead of just likeliness in neurons. Also, bijbij remains low if the activation uiui of capsule ii is low since û j|i length is proportional to uiui. i.e. bij should remain low between the mouth capsule and the face capsule if the mouth capsule is not activated.

The coupling coefficients cij is computed as the softmax of bij:

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

To make bij more accurate , it is updated iteratively in multiple iterations (typically in 3 iterations).

$$b_{ij} \leftarrow b_{ij} + \hat{u_{j|i}} \cdot v_j$$

Here is the final pseudo code for the dynamic routing:

Procedure 1 Routing algorithm.

1: **procedure** ROUTING($\hat{\mathbf{u}}_{j|i}, r, l$)
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: $\mathbf{c}_i \leftarrow \texttt{softmax}(\mathbf{b}_i)$
5: for all capsule j in layer $(l+1)$: $\mathbf{s}_j \leftarrow \sum_i c_{ij}\hat{\mathbf{u}}_{j|i}$
6: for all capsule j in layer $(l+1)$: $\mathbf{v}_j \leftarrow \texttt{squash}(\mathbf{s}_j)$
7: for all capsule i in layer l and capsule j in layer $(l+1)$: $b_{ij} \leftarrow b_{ij} + \hat{\mathbf{u}}_{j|i}.\mathbf{v}_j$
return $\mathbf{v}_j$

Loss

To allow multiple classes, margin loss has been minimized.

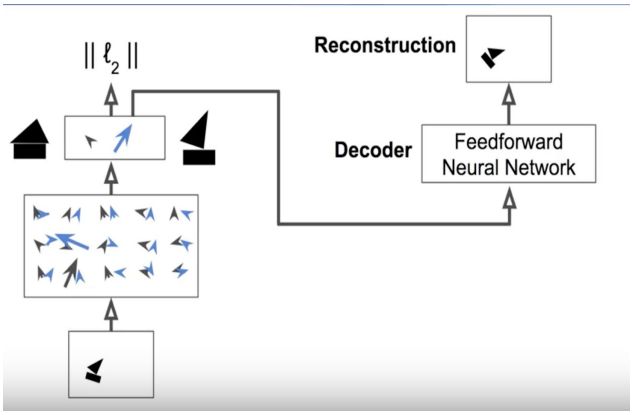
$$L_k = T_k \ \max(0, m^+ - ||\mathbf{v}_k||)^2 + \lambda \ (1 - T_k) \ \max(0, ||\mathbf{v}_k|| - m^-)^2$$

In the paper, values used are: m-= 0.1 ; m+=0.9 ; λ= 0.5

The margin loss essentially means that if an object of class k is present, then $||V_k||^2$ should be no less than 0.9. If not, then $||V_k||^2$ should be no more than 0.1. The sum is calculated for all the classes k.

Total Loss = Margin Loss + α Reconstruction Loss

In the paper α = 0.0005 (Scaled down so that Margin Loss Dominates).



Second loss used is the reconstruction loss which is the squared difference between the restructured image and the input image. Reconstruction loss forces the network to preserve all the information needed to reconstruct the image. It also acts as a regulariser which helps prevent overfitting and generate results when new data is fed.

Capsule Architecture

A simple CapsNet architecture is shown in Figure. The architecture is shallow with only two convolutional layers and one fully connected layer. Image is feed into the ReLU Conv1 which is a standard convolution layer. It applies 256 9x9 kernels to generate an output with 256 channels (feature maps). With stride 1 and no padding, the spatial dimension is reduced to 20x20. (28-9+1=20). This layer converts pixel intensities to the activities of local feature detectors that are then used as inputs to the *primary* capsules.

It is then feed into PrimaryCapsules which is a modified convolution layer supporting capsules. The primary capsules are the lowest level of multi-dimensional entities and, from an inverse graphics perspective, activating the primary capsules corresponds to inverting the rendering process. This is a very different type of computation than piecing instantiated parts together to make familiar wholes, which is what capsules are designed to be good at.

The second layer (PrimaryCapsules) is a convolutional capsule layer with 32 channels of convolutional 8D capsules (*i.e.* each primary capsule contains 8 convolutional units with a 9 × 9 kernel and a stride of 2). PrimaryCapsules uses 9x9 kernels with stride 2 and no padding to reduce the spatial dimension from 20x20 to 6x6 ($\lfloor 20-92 \rfloor+1=6 \lfloor 20-92 \rfloor+1=6$).It generates a 8-D vector instead of a scalar. Each primary capsule output sees the outputs of all 256 × 81 Conv1 units whose receptive fields overlap with the location of the center of the capsule. In total PrimaryCapsules has [32 × 6 × 6] capsule outputs (each output is an 8D vector) and each capsule in the [6 × 6] grid is sharing their weights with each other. One can see PrimaryCapsules as a Convolution layer with Eq. 1 as its block non-linearity.

It is then feed into DigiCaps which apply a transformation matrix Wij with shape 16x8 to convert the 8-D capsule to a 16-D capsule for each class jj (from 1 to 10).The final Layer (DigitCaps) has one 16D capsule per digit class and each of these capsules receives input from all the capsules in the layer below.

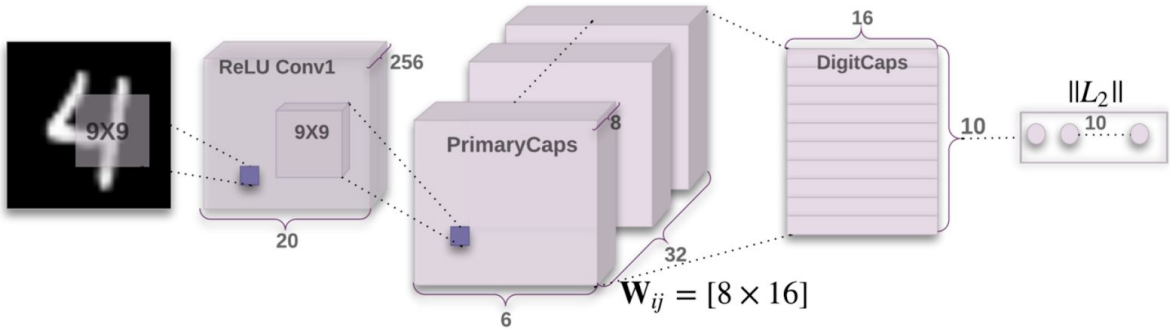


Figure 1: A simple CapsNet with 3 layers. This model gives comparable results to deep convolutional networks. The length of the activity vector of each capsule in DigitCaps layer indicates presence of an instance of each class and is used to calculate the classification loss. Wij is a weight matrix between each ui , i ∈ (1, 32 × 6 × 6) in PrimaryCapsules and vj,j ∈ (1,10).

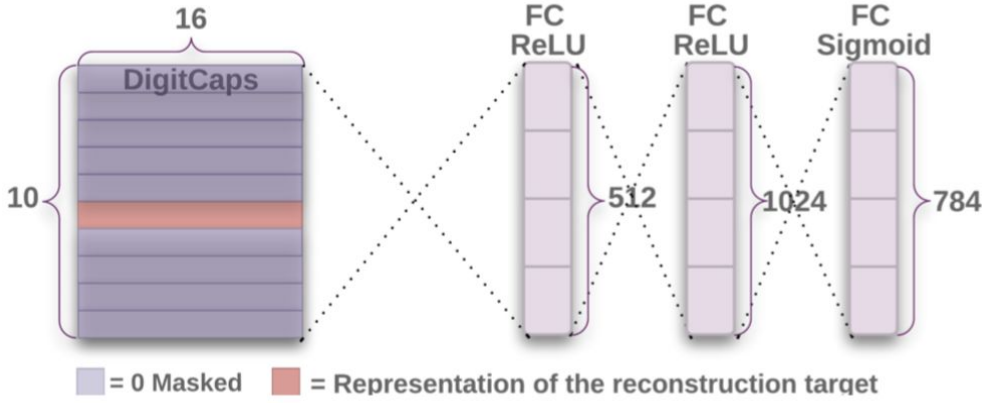
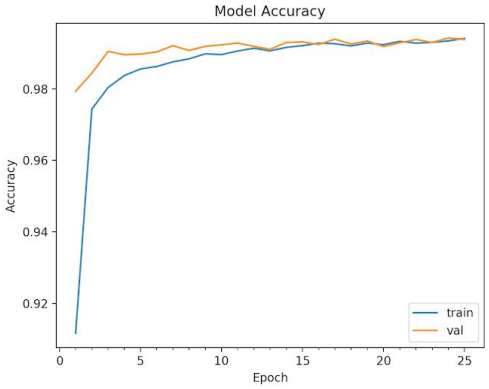


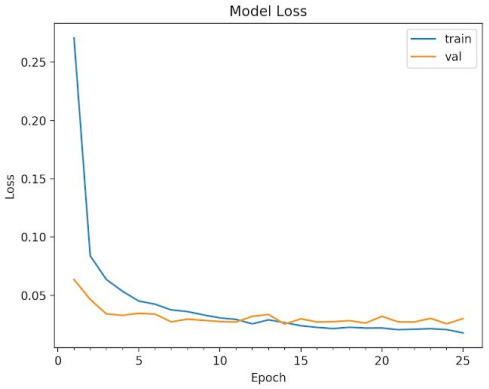
Figure 2: Decoder structure to reconstruct a digit from the DigitCaps layer representation. The euclidean distance between the image and the output of the Sigmoid layer is minimized during training. The true label is used as reconstruction target during training.

4. Experiments

A. MNIST

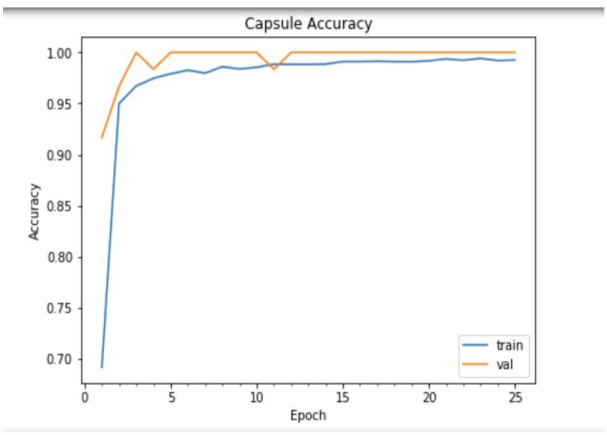
CNN
Epochs =25
Accuracy: 99.94%

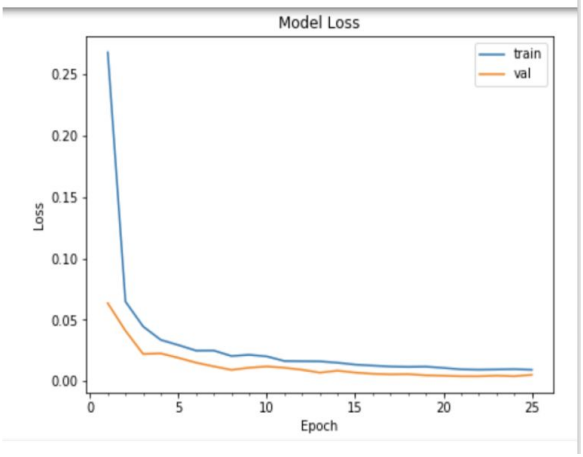




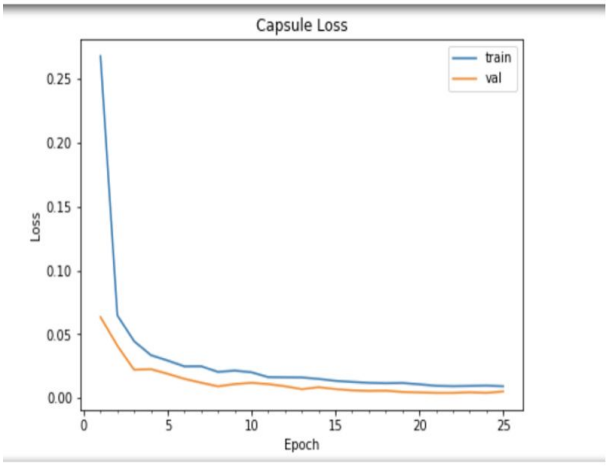
Capsnet

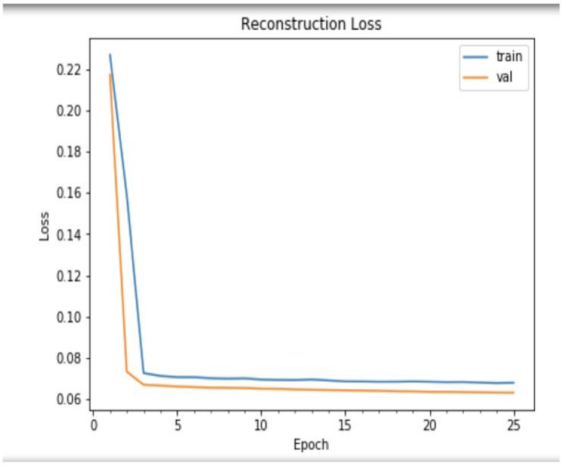
Epochs: 25
Fraction: 0.5
Training Accuracy 99.24% (increasing)
Validation Accuracy **100%**
Training Loss 0.0093565044475
Validation Loss 0.00521803





Loss

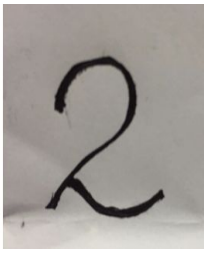




Marginal Loss **Reconstruction Loss**

Result on hand fed dataset

Sample data:



PreProcessing done:

- Conversion to Grayscale.
- Removing the edges to centralize the digit.
- Resize to 28 □ 28

	Original image(11)	**Inverted image(10)**
MNIST		
No preprocessing	6	10
Preprocessing	6	10
Capsnet		
No Preprocessing	8	8
Processing	5	4

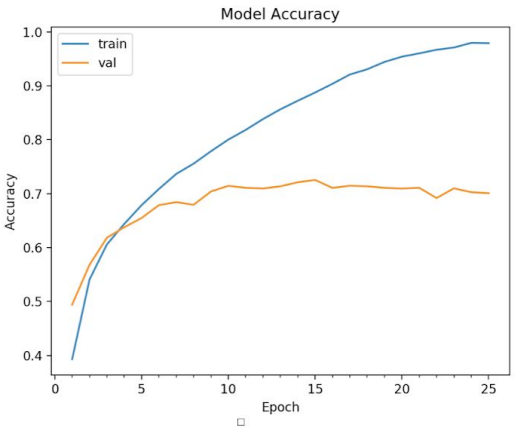
Observations:

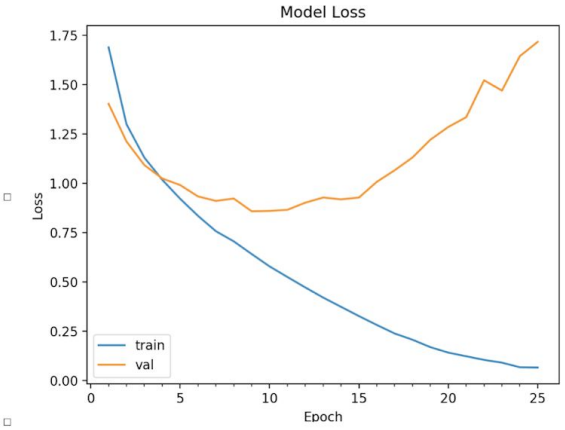
- Capsule network gives better accuracy for MNIST dataset.
- Inverting the images increases the number of digits recognised.
- Other preprocessing techniques result in improvement since the test images were already in the centre.

B. Cifar 10

CNN

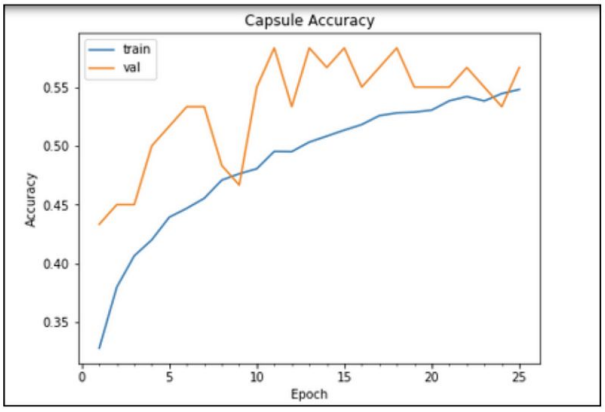
Epochs = 15
Training Accuracy: 97.31%
Validation Accuracy: 92.17%

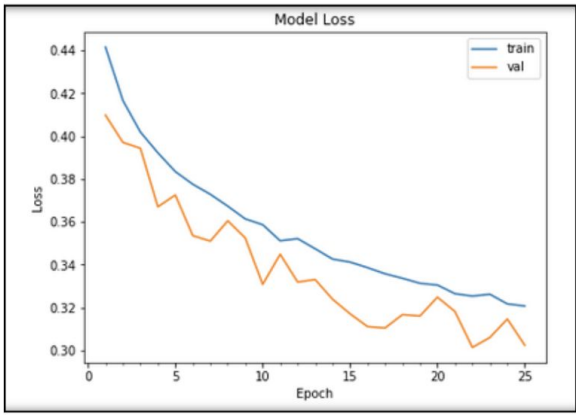


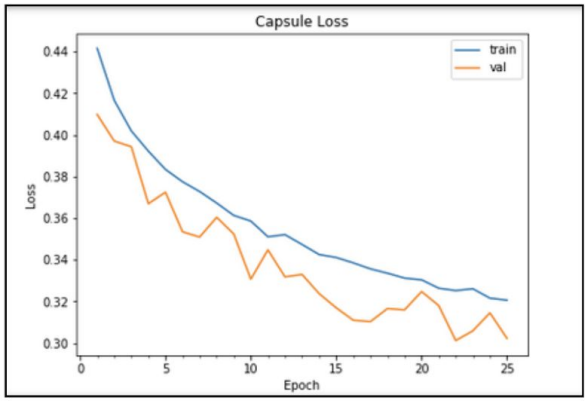


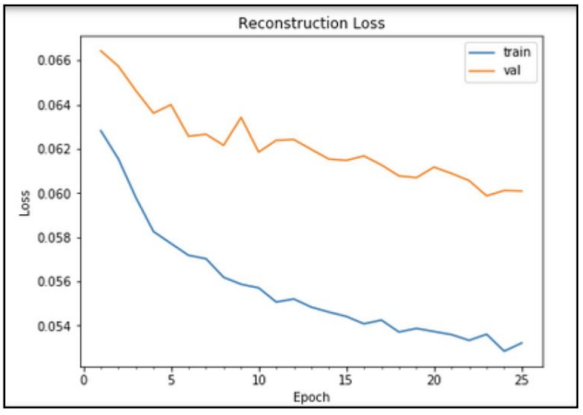
Capsnet

Epochs	25
Fraction	0.5
Training Accuracy	54.80% (increasing)
Validation Accuracy	56.66% (Max reached - 58.33 at epoch 10)
Training Loss	0.32 (decreasing)
Validation Loss	0.30



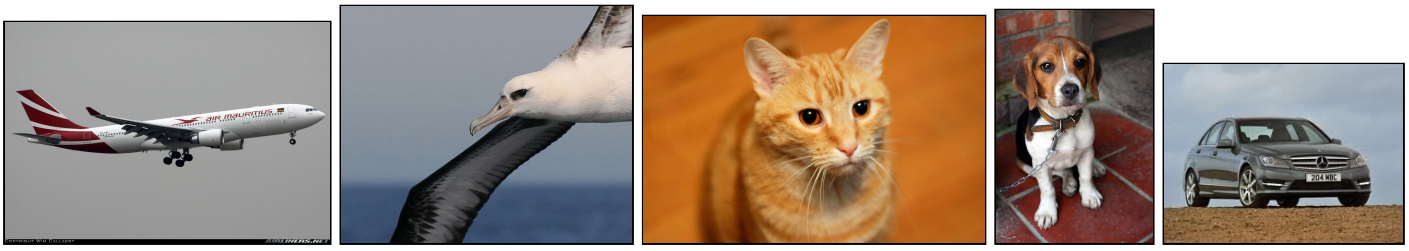






Result

Sample Data:



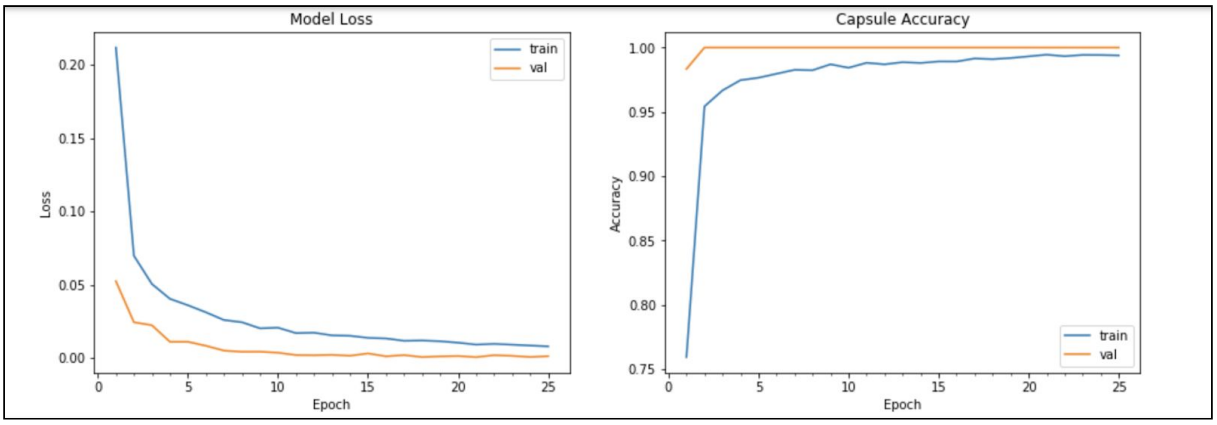
Class	**CN** **YCb**	**CNN- without YCbCr**	**CapsNet (epochs = 5)**	**Capsnet (25 epochs;0.5)**
Airplane (100)	81	37	12	38
Automobile(100)	73	51	9	11
Bird(100)	35	36	42	27
Cat(100)	29	10	4	9
Deer(29)	15	5	5	5
Dog(100)	29	15	3	9
Frog(10)	0	6	1	0
Horse(100)	47	28	2	7
Ship(100)	36	42	1	19
Truck(15)	8	10	0	3

Observations:

- Converting test images to YCbCr gives significantly better results in case of classification of most classes.
- Capsule networks does not give good results on the hand-fed data.
- Cifar 10 is a complex dataset. Capsule Network is unable to capture the complex features of the dataset accurately. The architecture of capsule needs to be modified in order to accommodate the complexity of the images.

C. Reducing the training data

MNIST dataset has 60,000 training images. We trained the capsule network with half of it i.e. 30,000 training images(3000 per class). The accuracy model loss obtained are:



Observations:

- The accuracy remains the same at 100%

Hence, we can say that Capsule network remains effective even when less data is available.

5. Conclusion:

Pros of Capsnet Architecture:

1. Reaches high Accuracy on MNIST data
2. Requires less training data
3. Position and pose Information are preserved (equivariance)
4. Routing with agreement is great for Overlapping objects
5. Capsule Activation nicely maps the hierarchy of parts
6. Offers robustness to affine transformations
7. Activation vectors are easier to interpret (rotation,thickness,skew etc)
8. Promising on Image Segmentation and Object Detection

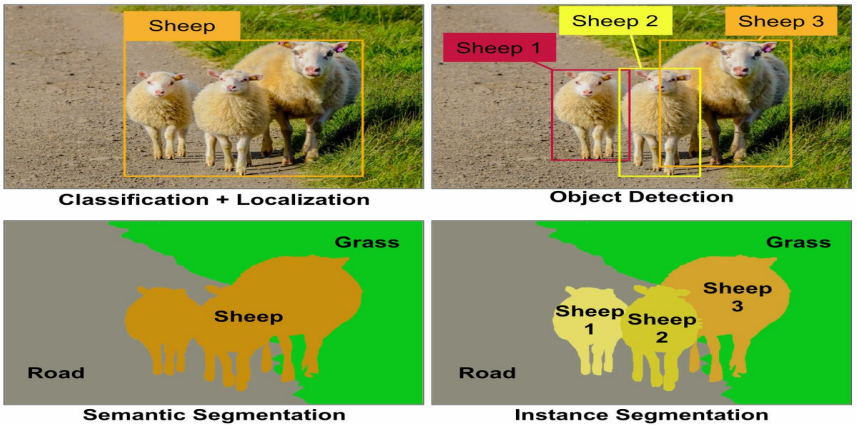
Cons of Capsnet Architure:

- Not a good state of art on cifar 10
- Not tested on even larger images like ImageNet
- Slow training, due to the inner loop (in the routing by agreement algorithm)

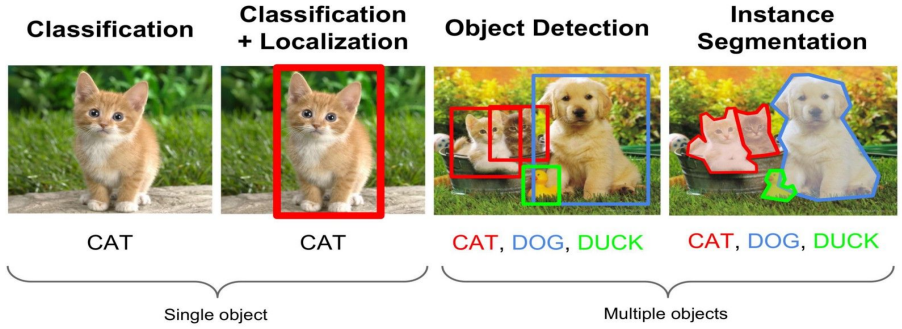
6. Object Detection

a. Literature Survey

Object Detection related Computer Vision Problems:



Object Detection:



Standard Data-set: Pascal VOC

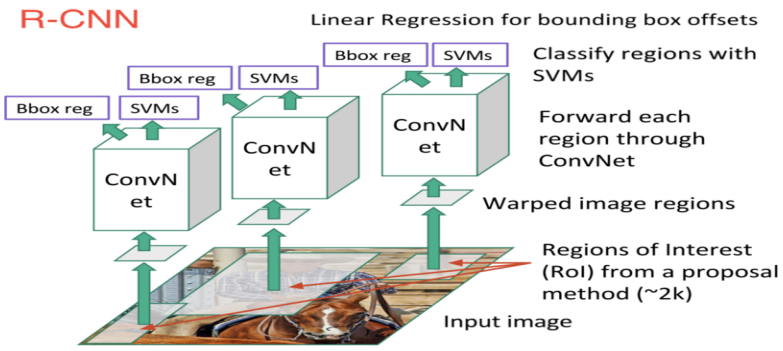
Classical Approach:

- Histogram of Oriented Gradients(HOG) features and Support Vector Machine (SVM) for classification.
- Shortcoming: Very Slow and Less Accuracy

Deep Learning Approaches:

1. *Brute Force Approach:* Run classification on all the sub-windows formed by sliding different sized patches(to cover each and every location and scale) all through the image. Computationally very Expensive.

2. RCNN:



R-CNN, or Region-based Convolutional Neural Network, consisted of 3 simple steps:

1. Scan the input image for possible objects using an algorithm called Selective Search, generating ~2000 region proposals which are basically tentative object locations and extents on the basis of local colour,rgb etc. No supervised learning and thus class-agnostic.
2. Run a convolutional neural net (CNN) on top of each of these region proposals.
3. Take the output of each CNN and feed it into
   a. an SVM to classify the region as background or object AND
   b. a linear regressor to tighten the bounding box of the object, if such an object exists.

Some Training Features:

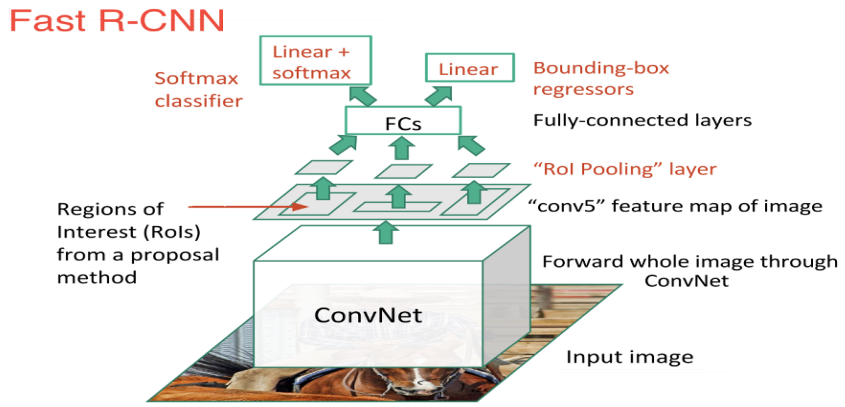
- Threshold for partial objects in Proposed Regions (IOU>0.5)
- Localization training is done independent of classification training.
- Classification training: SGD for end-to-end training
- Localization training: Minimizing the error of predicted coordinates (bounding boxes) against the ground truth coordinates. For this, linear regression layer is optimized on top of Conv5 layer after fine-tuning the network for classification.

Shortcoming: RCNN had to pass all the ~2000 regions from Selective Search(SS) independently through CNN and is therefore a very slow algorithm.

3. Fast-RCNN

- Fast RCNN tackles the downsides by installing the net with the capacity to back-propagate the gradients from FC layer to convolution layers.
- Second major change is a single network with two loss branches pertaining to soft-max classification and bounding box regression.
- These two changes reduces the overall training time and increases the accuracy in comparison to SPP net because of end to end learning of CNN.

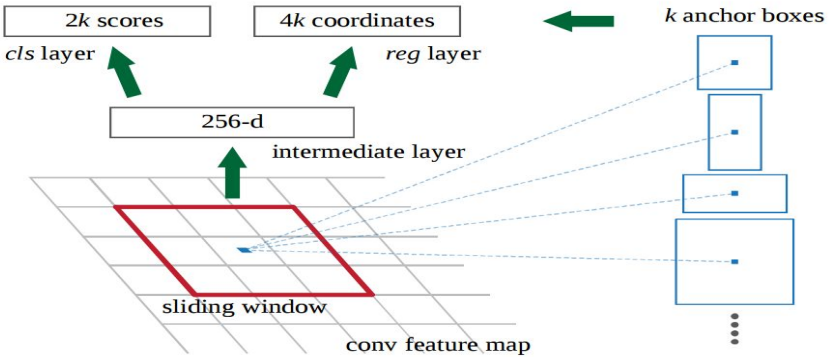
Shortcoming: Even though the Fast-RCNN performs much better in terms of speed,the bottleneck still remains the selective search algorithm for generating region proposals.



4. Faster R-CNN

Introduced the **Region Proposal Network (RPN):**

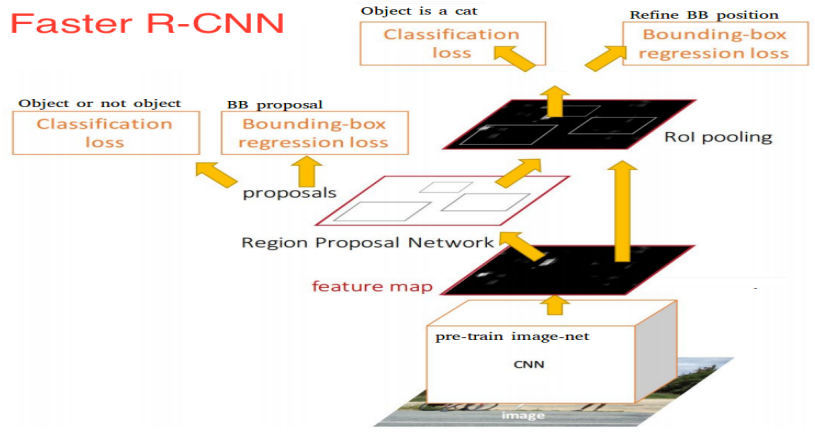
- At the last layer of an initial CNN, a 3x3 sliding window moves across the feature map and maps it to a lower dimension (e.g. 256-d)
- For each sliding-window location, it generates *multiple* possible regions based on *k* fixed-ratio **anchor boxes** (default bounding boxes) (tall box,larger box, etc)
- Each region proposal consists of
  - an "objectness" score for that region (whether it contains an object or not)
  - 4 coordinates representing the bounding box of the region



The 2*k* scores represent the softmax probability of each of the *k* bounding boxes being on "object." Although the RPN outputs bounding box coordinates, it does not try to classify any potential objects: its sole job is still proposing object regions. If an anchor box has an "objectness" score above a certain threshold, that box's coordinates get passed forward as a region proposal.

Once we have our region proposals, we feed them straight into what is essentially a Fast R-CNN. We add a pooling layer, some fully-connected layers, and finally a softmax classification layer and bounding box regressor. In a sense,

Faster R-CNN = RPN + Fast R-CNN.

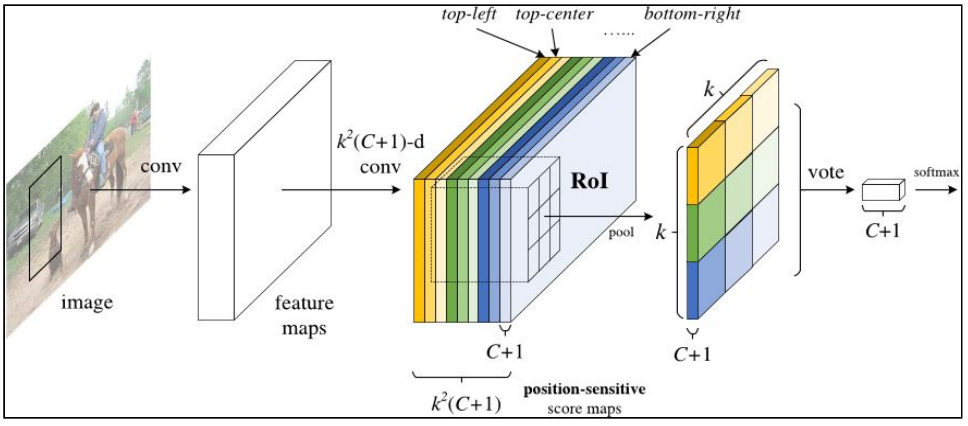


5. R-FCN(Region based Fully Convolutional Net):

Intuition behind R-FCN:

- Motivation behind the R-FCN: *increase speed by maximizing shared computation.*
- Fully convolutional: shares 100% of the computations across every single output
- Problem to fully convolutional: Trade-off between location invariance (for classification) and location variance(for bounding box)
- R-FCN's solution: **position-sensitive score maps**.
- Each position-sensitive score map represents *one relative position* of *one object class*. For example, one score map might activate wherever it detects the *top-right* of a *cat*. Another score map might activate where it sees the *bottom-left* of a *car*. Essentially, these score maps are **convolutional feature maps that have been trained to recognize certain parts of each object**.

Working of R-FCN:



1. Run a CNN (in this case, ResNet) over the input image
2. Add a fully convolutional layer to generate a **score bank** of the aforementioned "position-sensitive score maps." There should be $k^2(C+1)$ score maps, with k^2 representing the number of relative positions to divide an object (e.g. 3^2 for a 3 by 3 grid) and C+1 representing the number of classes plus the background.
3. Run a fully convolutional region proposal network (RPN) to generate regions of interest (RoI's)
4. For each RoI, divide it into the same k^2 "bins" or subregions as the score maps
5. For each bin, check the score bank to see if that bin matches the corresponding position of some object. For example, if I'm on the "upper-left" bin, I will grab the score maps that correspond to the "upper-left" corner of an object and average those values in the RoI region. This process is repeated for each class.
6. Once each of the k^2 bins has an "object match" value for each class, average the bins to get a single score per class.
7. Classify the RoI with a softmax over the remaining C+1 dimensional vector

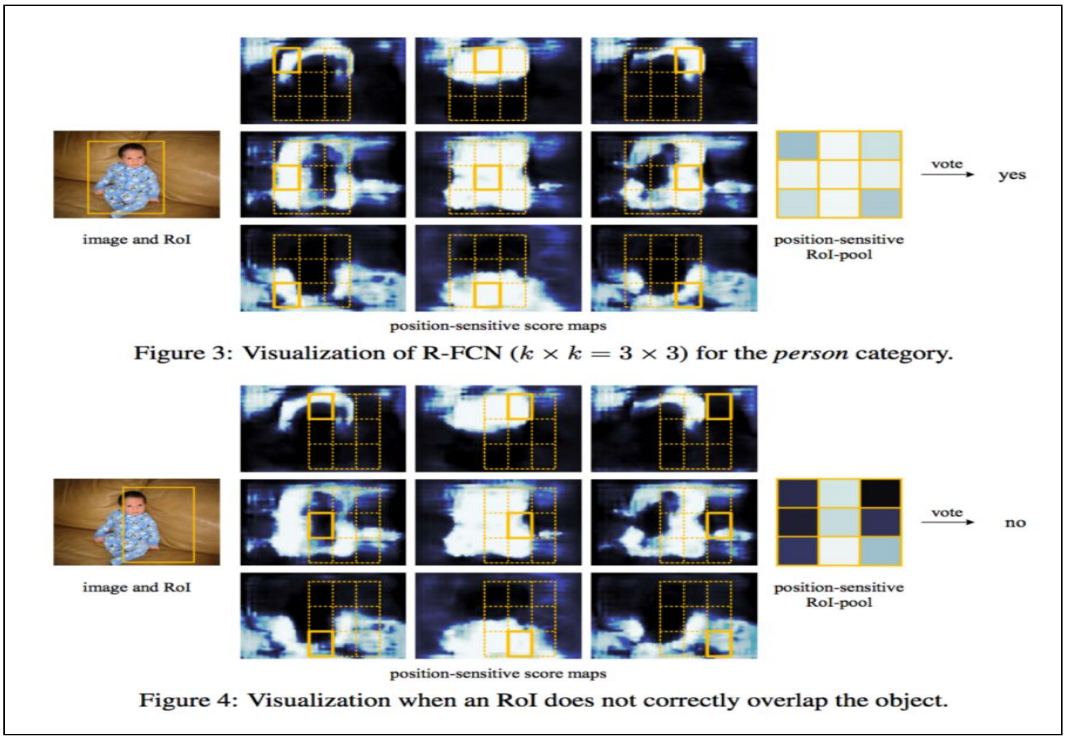


Figure 3: Visualization of R-FCN ($k \times k = 3 \times 3$) for the *person* category.

Figure 4: Visualization when an RoI does not correctly overlap the object.

With this setup, R-FCN is able to simultaneously address *location variance* by proposing different object regions, and *location invariance* by having each region proposal refer back to the same bank of score maps.R-FCN is several times faster than Faster R-CNN, and achieves comparable accuracy.

Final Analysis:

Deep learning techniques have evolved over time and there is a constant trade-off of between accuracy and time for computation.

Faster-RCNN, R-FCN and SSD(Single-Shot Detector) give comparable accuracies whereas R-FCN and SSD outperform Faster RCNN in terms of speed of computation.

7. Future Work

We propose to merge the feature generating capacity of CNN Variants with the Dynamic Routing Algorithm of the capsules to improve the final predicting accuracy in Image Segmentation and Object Detection techniques. R-FCN object detection architecture can make use of dynamic routing between the score maps generated instead of the ROI pooling to detect objects. Also Capsnet accuracy on overlapping objects is still not tested.

8. References

1. https://arxiv.org/abs/1710.09829
2. https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9
3. https://medium.com/ilenze-com/object-detection-using-deep-learning-for-advanced-users-part-1-183bbbb08b19
4. https://tryolabs.com/blog/2017/08/30/object-detection-an-overview-in-the-age-of-deep-learning/
5. https://jhui.github.io/2017/11/03/Dynamic-Routing-Between-Capsules/
6. https://www.oreilly.com/ideas/introducing-capsule-networks
7. https://www.youtube.com/watch?v=pPN8d0E3900&t=244s
8. Papers:
   Dynamic Routing between Capsule
   https://arxiv.org/pdf/1710.09829.pdf
   R-FCN:
   https://arxiv.org/pdf/1605.06409.pdf
   SSD:
   https://arxiv.org/pdf/1512.02325.pdf
   Fast-RCNN:
   https://arxiv.org/pdf/1504.08083.pdf
   Faster-RCNN:
   https://arxiv.org/pdf/1506.01497.pdf