Compositionality in Deep Learning Networks

Nithin K Guruswamy

***Abstract*—Compositionality is observed everywhere in nature. The PART-A of the paper explores the concept of compositionality in the context of Deep learning taking motivations from biology (Brain) and solves problems in deep learning computation efficiently. The PART-B explores problems in computer vision tasks such as object detection. Using the core idea of compositionality inspired from the biological visual cortex of the brain, solves the object detection task in Deep neural networks.**

Index: Modular Neural network. Deep neural network (DNN), Expectation maximization algorithm, Convolution neural network.

I. Introduction and Motivation:

There are several motivations for studying compositionality/modularity in case of Deep Neural network. It can be Computational, Biological, Psychological, and Implementational. In this paper I will mostly be exploring the biological motivations. Large scale neural networks or deep Neural networks give good prediction performance for big training data. As shown in the Figure 1 the more amount of training data you feed to the Deep neural networks, it provides proportional linear performance increase as amount of input training data increases. Whereas, other neural networks like medium and small neural networks does provide linear performance but, flattens out for much lesser sized training data. The other non-Neural network training algorithms like SVM, Logistic or linear regression gives even less performance as they flatten off much earlier ,thus deep neural networks are currently the best performing machine learning algorithms with the ability to train on large training data. However, this large capacity comes with a price of huge computing complexity/cost. The computation of the activation functions from its weights from all its deep layers and weight calculations using backpropagation etc., entails a huge compute cost. Also, the training time to train such complex algorithm is high.

We know that a Deep Neural network is a universal function approximator. For a given machine learning problem, we currently do not know how many layers of Deep neural network are enough to approximate it well, so that DNN performs well with less generalization error and with low compute cost and low training time.

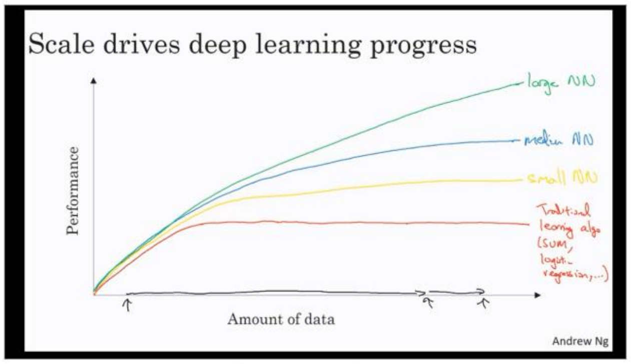


Fig. 1. Performance of some of the Machine learning algorithms vs. data size

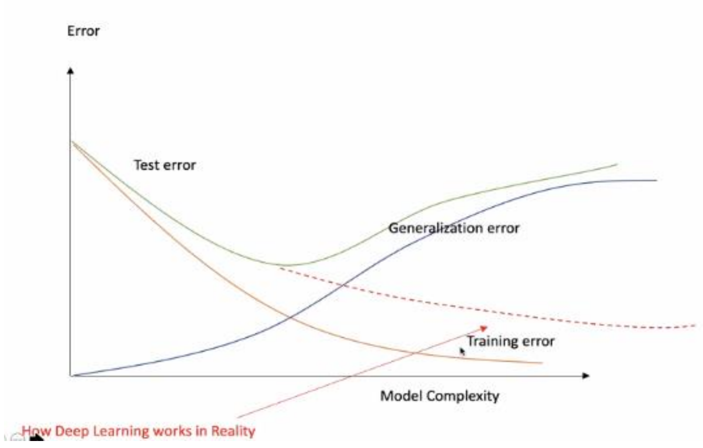


Fig. 2. Shows the Test and training errors decrease as more hidden layers are added into Deep neural network but it also the generalization errors.

As shown in the figure 2: the performance increase (test and training errors decrease) as the Neural network gets deeper with more complexity by adding more hidden layers. However, it also increases the generalization error.

In the PART-A of the paper, Modular neural network (MNN) tries to solve this problem by being inspired by the biological brain using compositionality. Deep neural network can be thought of as composed of multiple smaller neural networks modules. Modular neural networks decompose the complex monolithic neural network problem into modules like different regions of brain where, each region specializes in a diverse task. By decomposing the deep neural network into modules, it achieves many advantages.

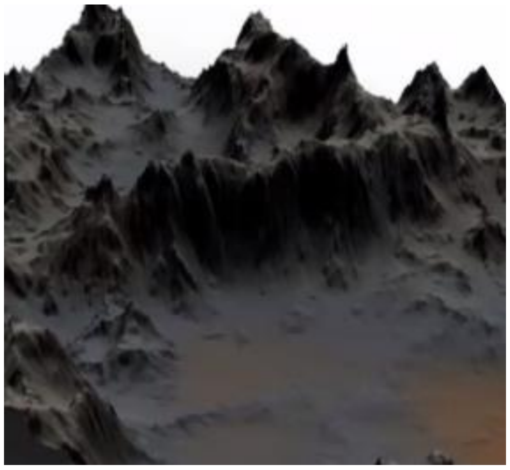


Fig. 3. 3-D visualization of a non-convex cost function of a Deep neural networks which requires complex optimizations which MNN avoids

First, the Modular neural network avoids the non-convex complex optimization problems as shown in Figure 2 encountered by monolithic neural network without decreasing the learning capacity and by using an ensemble of smaller Neural network modules. The smaller Neural network modules can be optimized easily as mostly it will be a convex function rather than a non-convex one for the monolithic deep neural network.

Secondly, MNNs enable to use a priory trained module to be used with other modules. This enables transfer learning and decreases the time taken to train a deep neural network. Also, the modular neural network is immune to temporal and spatial crosstalk – a problem faced by Deep neural network during training. Finally, theoretical and other empirical analysis shows MNN yields better generalization than its constituent modules.

Overall, modularization in MNNs leads to efficient and less complicated computation as MNNs do not suffer from high coupling burden as in monolithic deep neural network. This makes modular neural network scalable to implement a large deep neural network efficiently. Also, as an inspiration taken from biological human brain, only critical parts of the brain are active and unused regions are switched off to conserve energy due to modularity.

Taking inspiration from all the above observation, a new way of automatically decomposing the Monolithic modular neural network into reusable modules is being proposed. The choice of module selected is modeled as a latent variable in a probabilistic model. It learns the decomposition and module parameters end to end by maximizing a variational lower bound of the likelihood. A small fixed number out of a larger set of modules is selected to process a given input, and only the gradients of these selected modules need to be calculated during the backpropagation. This novel method is known as Expectation maximization (EM) algorithm and gives good module selection diversity compared to other modular Neural network algorithms. The latter part also explores the experimental verification of this method over other existing modular neural computation methods.

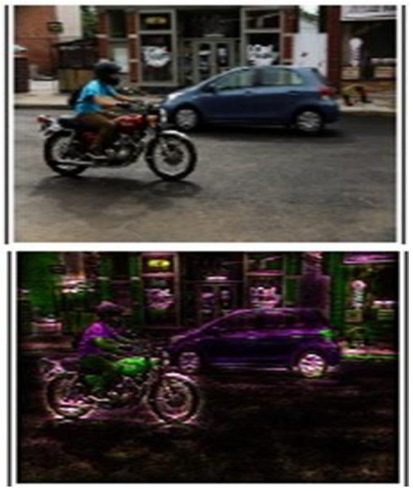


Fig. 4. Top half image is being seen as image in bottom half after being generalized by VGG-16 CNN after sufficient training.

The PART-B of the paper discusses the application of compositionality on Convolution Neural network (CNN) - which is a type of Deep neural network having 2-dimensional convolution as its activation function. We also observe the property of compositionality in nature especially in biological systems like in visual cortex of brain. The visual cortex of the brain processes the image information and constitutes biological vision. The area of the visual cortex that processes the object-scene interaction part exhibits non-linear compositionality. CNN is also able to model biological vision and have proved their competence in many visual processing tasks like image classification, object class detection, instance segmentation, image captioning and scene understanding. Since Convolution neural network is able to model biological vision shown in Figure 4., it also exhibits nonlinear compositionality and it is able to model human-object interaction well.

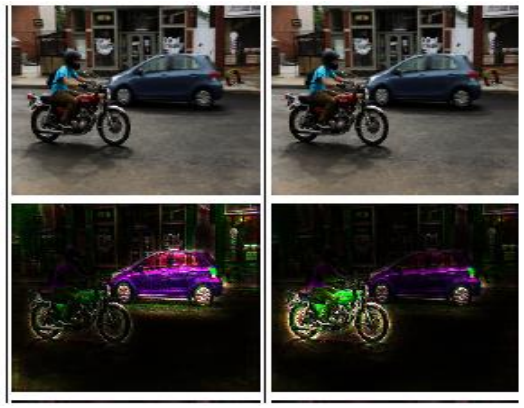


Fig. 5. CNN is not able to detect/select only object from its background scene like car or motorcycle.

However, CNN is not able to detect object from other non-contextual objects or background scene as shown in Figure 5. To improve the detection of a familiar object from a background context we need to inculcate linear compositionality to the CNN. It is a known fact that even the brain has a separate region for processing the object and for background scene. Also, the brain can exhibit linear compositionality. Linearity in CNNs can be inculcated by novel training method by applying separate masks to the objects and background and using separate CNNs for training and that will be discussed in much more detail later in the subsequent sections. Finally, an extensive experimental evaluation is performed to verify that this new novel training method does give an improved object detection from the scene by objective and subjective testing.

II. PART -A

A. *Modular Neural network using Expectation Maximization (EM) algorithm:*

1) *Deep Neural networks:*

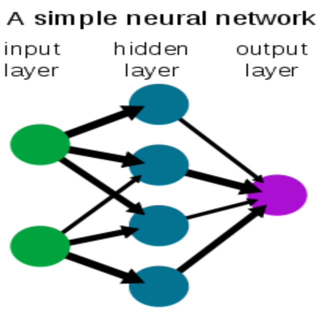


Fig. 6. Shows a Neural network with a single hidden layer.

The Neural network shown in figure 6 has only 1 hidden layer. It has input layers, output layer and hidden layers with many interconnections having weights. The input is multiplied by the weights and undergoes transformation as per the activation function. The output of one layer is the input to the subsequent layer. This flow from left to right is known as forward propagation.

During training phase, the output of the Neural network is calculated using forward propagation. Error or cost function is calculated by taking difference of actual output (in training set) to the output of neural network. This error is minimized by iteratively adjusting the weights to reduce the cost or error known as backpropagation. A Simple neural network is able to approximate simple functions and its cost function will be mostly convex and many simple optimization techniques like Stochastic gradient descent will be able to able to able to optimize the cost function to its global minimum.

A Deep neural network contains many hidden layers as shown in Fig. 7. A deep neural network can approximate many complex functions. However, as stated previously it usually has complex non-convex optimization problem and error or cost function is not reduced fully thus, Learning may be not be efficient and generalization drops.

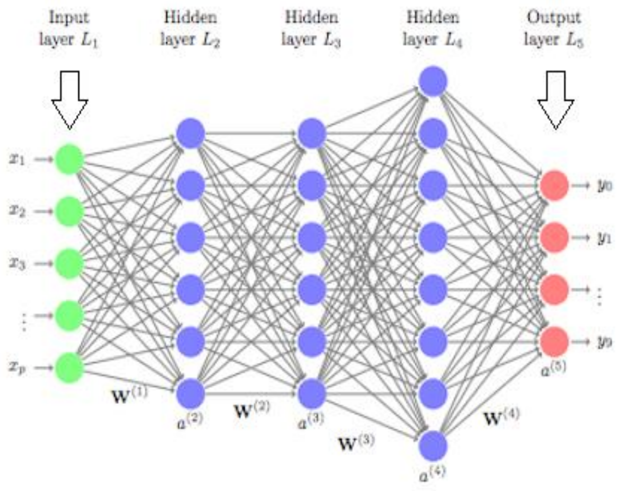


Fig. 7. Shows a deep neural network with many hidden layers which can be considered as composed of many simple neural network shown in Fig.6

Hence compositionality in deep learning networks by the way of Modular networks, which splits the deep networks into smaller ones, which has easier convex optimization functions can help here.

2) EM algorithm:

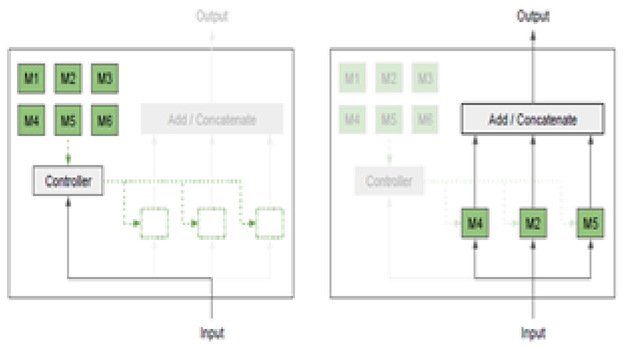


Fig. 7 . (a) Shows a Modular neural network of modules M=6;(b) shows K=3 modules are selected out of M=6 .

Consider a modular neural network layer of having M=6 modules at each hidden layer L and K=3 random modules are selected by a controller to process a given input. The backpropagation is applied to only these 3 modules only.

Let Modular Deep Neural Network be composed of *i* modules where $f_i(x, \theta_i)$ is the function modeling an *i*th module, which takes x as input vector and θ_i are the parameters for the *i*th module.

Let l be a modular layer,$l \in \{1, \dots, L\}$, be composed of M modules. The controller selects k=3 modules out from M modules.

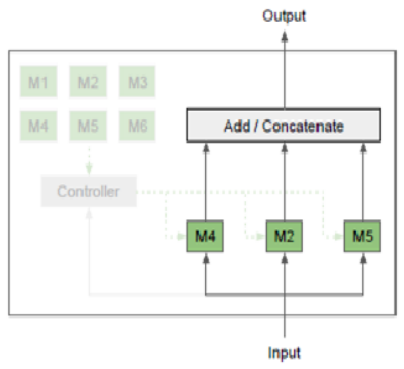


Fig. 8. Shows a Modular NN with k=3 modules (M4, M2 and M5) selected.

Let $a^{(l)}$ be the random variable denoting indices of module selected.

$$a^{(l)} \in \{1,2,..,M\}^k$$

If M=6 ; K=3 then

$$a^{(l)} \in \{1,..,215\}$$

Let the controller distribution at layer *l* be $p(a(l)|x(l), \emptyset(l))$

is parameterized by $\emptyset(l)$ where $\emptyset(l)$ is the parameter of distribution which is also dependent on input $x(l)$

The DNN output at level *l* from the module is given by summation method:

$$y^{(l)} = \sum_{i=1}^{k} f_{a_i^{(l)}}\left(x^{(l)}; \theta_{a_i^{(l)}}\right) \rightarrow (1)$$

Where $y(l)$ can form input for +1 layer

The controller module selection at all layers has a joint distribution

$$P(a|x, \phi) = \prod_{l=1}^{L} P\left(a^{(1)}\middle|x^{(l)}, \emptyset^{(l)}\right) \rightarrow (2)$$

The entire DNN composed of 'a' module can be used to parameterize the distribution over the final Neural network output layer

$$y \sim P(y|x, a, \theta).$$

The joint distribution of MDNN over output y and module selection 'a' is

$$P(y|x, a, \theta) = P(y|x, a, \theta)P(a|x, \emptyset) \rightarrow (3)$$

Selection of module is random so 'a' is stochastic.

Since Module selection by controller is stochastic, so 'a' is considered as a latent variable, output is defined as

$$P(y|x, a, \theta) = \sum . P(y|x, a, \theta)P(a|x, \emptyset) \rightarrow (4)$$

Selecting K modules at each of L layer makes the 'a' have M^{kL} states and summation of equation (4) is difficult.

For a given collection of input and output data $(x_n, y_n), n = I_1 \dots, N$,from a probabilistic model, the training objective is maximum likelihood, that is we try to adjust module parameter θ and control parameter ϕ to maximize log likelihood.

$\mathcal{L}(\theta,\emptyset) = \sum_{n=1}^{N} log\ \ P(y_n|x_n,\theta,\phi)$ $\rightarrow$ (5)

Now to evaluate the summation over states of ‘a’ in Equation (4)

$$P(y|x,\theta,\emptyset) = \sum_{a} P(y|x,a,\theta)\,P(a|x,\phi)$$

we use generalized expectation maximization for single datapoint to ward off difficulty in calculating summation over latent variable ‘a’, when we use it in Equation (5) for a single data point we get

$log\ P(y|x,\theta,\emptyset)$ >= - $\sum_a q(a)\ log\ q(a)$ + $\sum_a q(a)\ log\big(p(y|x,a,\theta)p(a|x,\phi)\big)\ \ \equiv\ L(q,\theta,\phi)$ $\rightarrow$ (6)

Where $q(a)$ is a variational distribution used to tighten the lower bound of likelihood.

We can compactly write Equation (6) as

$L(q,\theta,\emptyset) = E_{q(a)}[\log p(y,a|x,\theta,\emptyset]$ + H$[q]$

Where H$[q]$ is the entropy of the distribution q . We try to adjust $q,\theta,\emptyset$ to maximize L.

The Partial M-step on $(\theta,\emptyset)$ is defined by taking multiple gradient descent steps where the gradient is

$\nabla_{\theta,\emptyset} L(q,\theta,\phi)$ = $\nabla_{\theta,\emptyset} E_{q(a)}[log(p(y,a|x,\theta,\phi))]$ $\rightarrow$ (8)

In Practice we randomly select a minibatch of datapoints per iteration. Evaluating this gradient requires a summation over range of ‘a’.

To avoid this, Viterbi EM approach effective in which q(a) is constraint to the form

$$q(a) = \delta(a,a^*) \qquad \rightarrow (9)$$

Where $\delta(a_1 a^*)$ is the Kronecker delta function. This is 1 if x=y or 0 otherwise.

A full E-step (estimating latent variable ‘a’ for max likelihood) for a single datapoint would now update a^* to

$a^*_{new} = argmax_a\ P(y|x,a,\theta)p(a|x,\phi)$ $\rightarrow$ (10)

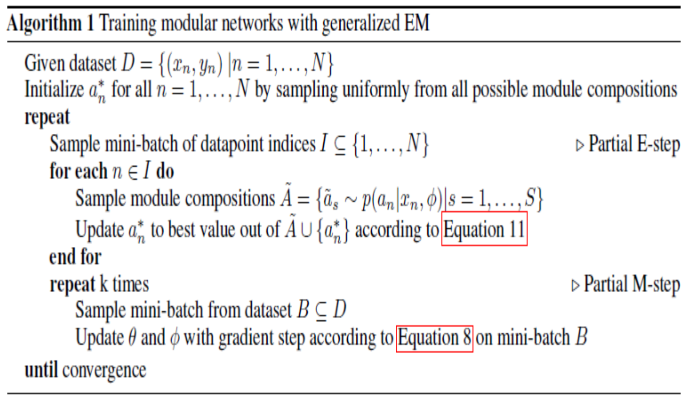
Where argmax is the maxima. For all datapoints, we make E-step partial in 2 ways for tractability.

1. We choose the best from only S samples
   $\widetilde{a_s} \sim P(a|x,\phi)$ for $s \in \{1,\ldots,S\}$
   Keep the previous a^* if L(likelihood) decreases

$a^*_{new} = argmax\ _{(a)} P(y|x,a,\theta)p(a|x,\phi)$ $\rightarrow$ (11)

Where $a \in \{\tilde{a}_{s|}s \in \{1,\ldots,S\}\}\cup\{a^*\}$

2. We apply this to a mini batch while keeping a^* associated with all other data points constant.



```
Algorithm 1 Training modular networks with generalized EM
  Given dataset $D = \{(x_n, y_n)\,|n = 1,\ldots,N\}$
  Initialize $a^*_n$ for all $n = 1,\ldots,N$ by sampling uniformly from all possible module compositions
  repeat
    Sample mini-batch of datapoint indices $I \subseteq \{1,\ldots,N\}$          ▷ Partial E-step
    for each $n \in I$ do
      Sample module compositions $\tilde{A} = \{\tilde{a}_s \sim p(a_n|x_n,\phi)|s = 1,\ldots,S\}$
      Update $a^*_n$ to best value out of $\tilde{A} \cup \{a^*_n\}$ according to Equation 11
    end for
    repeat k times                                                              ▷ Partial M-step
      Sample mini-batch from dataset $B \subseteq D$
      Update $\theta$ and $\phi$ with gradient step according to Equation 8 on mini-batch $B$
  until convergence
```

Fig. 9. Shows a summarized EM algorithm.

3) *Alternative Modular Neural Network Algorithms:*

a) Reinforce Method: This method maximizes the lower bound on log likelihood.

$B(\theta,\emptyset) \equiv \sum_a P(a|x,\phi)\ logP(y|x,a,\theta) \leq\ L(\theta,\phi)$

The gradients are obtained as below:

$$\nabla_{\phi} B(\theta,\emptyset) = E_{p(a|x,\phi)}(log\ P(y|x,a,\theta)\nabla_{\phi} logp(a|x,\phi))$$

$$\nabla_{\theta} B(\theta,\emptyset) = E_{p(a|x,\phi)}(\nabla_{\phi} logp(y|x,a,\theta))$$

The expectations are approximated by sampling from $p(a|x,\phi)$.

b) Noisy top-K mixture of experts: This method is based on mixture of experts. Mixture of experts is a weighted sum of parameterized function each having certain assigned weights. A gating network is used to predict weight of each expert as shown in Fig. 9.

This can help model compositionality by splitting functionality into multiple separate components with different weights.

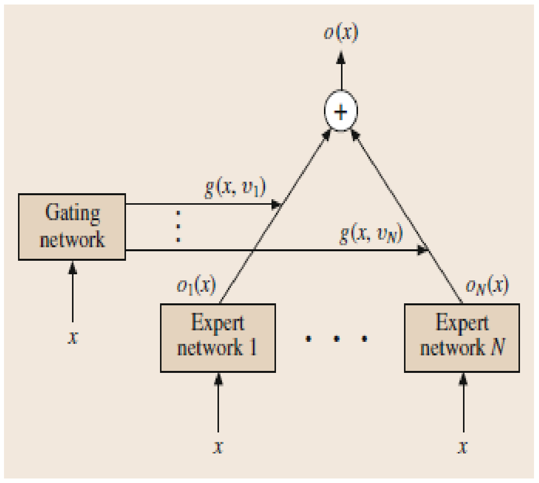


Fig. 9. Shows a mixture of experts

To disable any expert, set all non k weights to -infinity. Noise is added to the output of gating network. Only K units are evaluated and gradients backpropagated. However, there are many issues with these 2 alternate Modular neural network models like lack of diversity and high-test perplexity (lower is better).

4) *Module collapse:*

This is a phenomenon wherein, same modules are selected for different batches of Input B. This occurs because due to self-reinforcing effect, wherein favored modules are trained rapidly. Most of the Alternate modular methods like Reinforce and Noisy top-k mixture of experts suffered from this effect and used regularization. In regularization, the modular algorithm is forced to select different module within an input mini-batch. But regularization has its own set of performance problems. The EM algorithm does not suffer from Module collapse and no regularization is required.

5) *Testing methodology – test metrics*

Two test metrics are defined

a) *Module selection entropy:*

$$H_a = \frac{1}{BL}\sum_{l=1}^{L}\sum_{n=1}^{B}\mathbb{H}\left[p(a_n^{(l)}|x_n,\phi)\right]$$

Where B is the batch size. The module selection entropy should be small to achieve convergence. That is a given particular input batch being assigned to a module which gives high likelihood.

b) *Batch module selection Entropy:*

This is Module selection is done over entire batch.

$$H_b = \frac{1}{L}\sum_{l=1}^{L}\mathbb{H}\left[\frac{1}{B}\sum_{n=1}^{B}p(a_n^{(l)}|x_n,\phi)\right]$$

This metric should be high to achieve high diversity so that All different input batches are not assigned a single module.

Module collapse will have low Hb.

6) *Application on EM algorithms:*

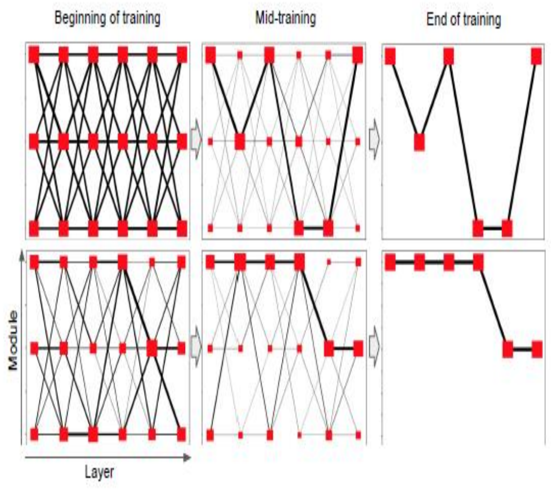


Fig. 10. Shows a summarized EM algorithm.

Consider a DNN with layers L=6, each layer has Modules M=3. K out of M modules are selected where K=1.

The Input task is image classification. Classification task begin with arbitrary assignment of modules for each input. Both upper and lower sections of the Figure 9, depict different subsets of input data points. Algorithm clusters similar inputs by assigning them same modules.

In each partial E-step we use controller $P(a^{(l)}|x^{(1)}, \emptyset^{(l)})$ as a guide to reassign modules to input. In each partial M-step module params θ and controller parameters ϕ are adjusted.

The EM algorithm assigns similar inputs to a common group of modules as shown in top and bottom parts of Fig 9

7) *Language Modelling using EM Algorithm:*

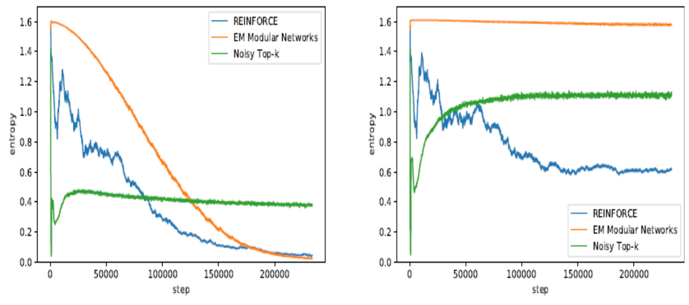
Modularization can be used in recurrent neural networks (RNNs) also. It is a known fact that the RNNs are good in modeling sequence to sequence tasks, that is tasks having temporal variation characteristics like speech. The modularization can be used to update state in RNN. This enables to predict modules based on context.

The modularization experiment is carried out on Gated recurrent unit (GRU) – which is a type of long short-Term Memory (LSTM) with no output gate. GRUs have input and forget gates only. The state update operation is modularized with only K modules are selected and gradients applied at each step. Penn treebank dataset is used which has around a million words with 10k vocabulary. Four module configurations are tested, where K/M = 1/5,3/5,1/15 and 3/15.

EM-approach is compared with unregularized reinforce method, noisy-top k and baseline Gated Recurrent unit. We can see that in Fig EM algorithm approach has lowest test perplexity (accuracy in predicting probability distribution) among other modularizing algorithms.

Table 1: Test perplexity after 50,000 steps on Penn Treebank

Type	#modules (M)	#parallel modules (K)	test perplexity
EM Modular Networks	15	1	**229.651**
EM Modular Networks	5	1	236.809
EM Modular Networks	15	3	246.493
EM Modular Networks	5	3	236.314
REINFORCE	15	1	240.760
REINFORCE	5	1	240.450
REINFORCE	15	3	274.060
REINFORCE	5	3	267.585
Noisy Top-k ($k = 4$)	15	1	422.636
Noisy Top-k ($k = 4$)	5	1	338.275
Baseline	1	1	247.408
Baseline	3	3	241.294



(a) Module selection entropy H_a (b) Batch module selection entropy H_b

Fig. 11. Shows Ha and Hb for Language modelling using Penn tree bank for EM, Reinforce and noisy top-k Mixture of experts

8) Image classification using EM algorithm in Convolutional neural network (CNN):

It is a known fact that CNN can perform well in computer vision tasks. Here modularization is applied in the context on CNN. Comparison is done for EM algorithm enabled feed forward neural network, EM modularized CNN (Both 2 fully connected layers and 3 convolution layers are modularized) and baseline CNN. For testing CIFAR-10 dataset is used.

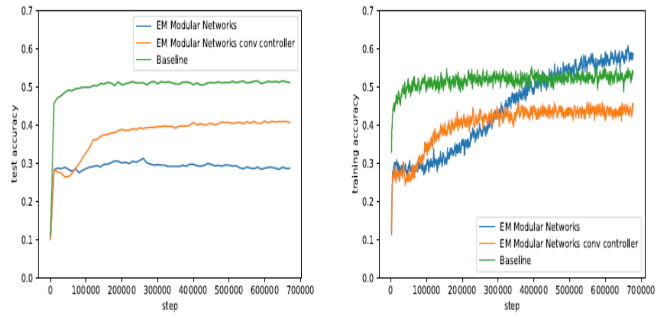


Fig. 11. Shows the testing and training accuracy vs step size

We can see that Modularized approach gives higher training accuracy and generalizes well due to low test perplexity scores.

III. PART -B

A. Compositionality in Deep neural network in computer vision application:

1) Introduction to Convolutional neural network (CNN):

CNN is a type of neural network which uses non-linear activation function such as 2D-convolution. Convolution Neural Network (CNN) is a type of edge detector and gives good performance in many computer visions tasks such as image classification, scene understanding and object detection. CNN models object-scene interaction as non-linear compositionality like brain.

2) CNN relation to visual cortex:

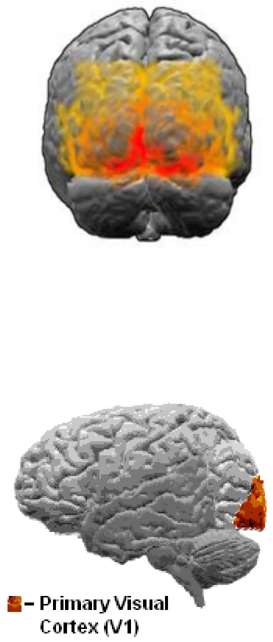


Fig. 12. Shows the Visual cortex region of the brain

The CNNs do take a biological inspiration from the Visual cortex. We know that Visual cortex is part of brain that processes vision. The visual cortex has small regions of cells that are sensitive to specific regions of the visual field. This idea was expanded upon by a fascinating experiment by Hubel et al in 1962 where they showed that some individual neuronal cells in the brain responded or fired only in the presence of edges of a certain orientation. We know that the CNN also is sensitive to edges in the images. Hence there is a similarity between CNN and biological vision processing part of the brain.

3) Convolution Neural network structure:

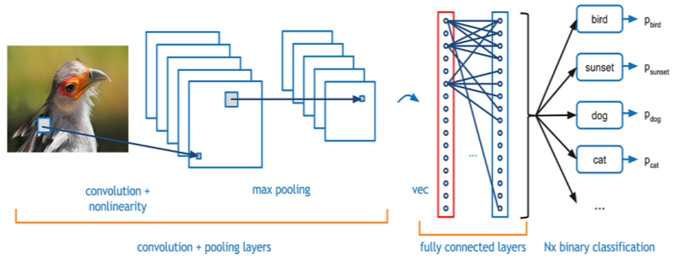
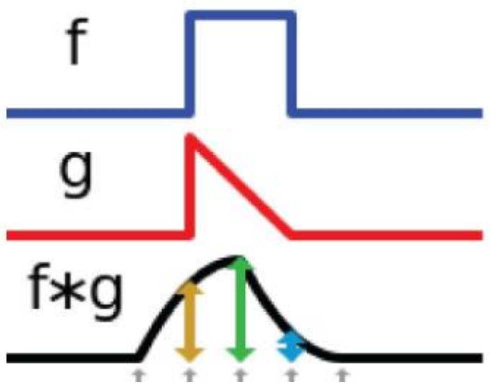


Fig. 13. Shows the Visual cortex region of the brain

As shown in the figure 13. CNN Consists of many alternate layers of Convolution and sub sampling (Pooling). Convolutions are done with a customized edge detecting kernel. Last layer is a fully connected DNN which does N-ary classification.



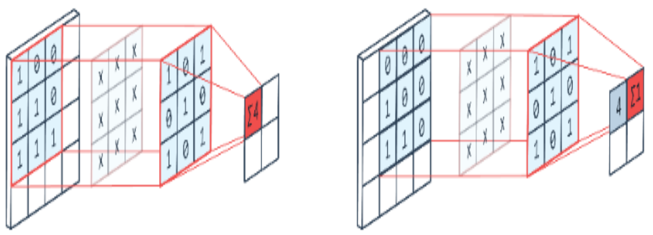


Fig. 14. Shows the 1D, 2D convolution operation

Convolution operation as shown in Fig 14. Looking at the 1D convolution we can see that it spreads the input signal based on the other signal.2D convolution in CNN is done with a specialized edge detecting kernel. After convolution, the dimension of output changes due spreading nature of the convolution operation.

4) Visual cortex shows non-linear compositionality:

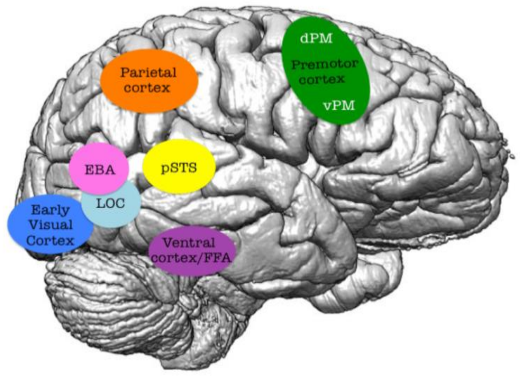


Fig. 14. Shows different parts of the visual cortex

The Downstream regions of the visual cortex process high-level visual information. There are specialized regions for processing which responds well to certain visuals stimulus .The LOC (Lateral Occipital cortex) responds on Objects, The

EBA (Extrastriate body area) on Human poses and The pSTS (posterior Superior temporal Sulcus) responds well on human interactions with objects. Certain areas in the brain like pSTS which are highly sensitive specifically to human-object interactions.

The neural representations of interactions are not a linear sum of the human and object representations. This is as per the inference of experiments done by Baldassano et al.

5) Experiment to check if CNN mimics visual cortex:

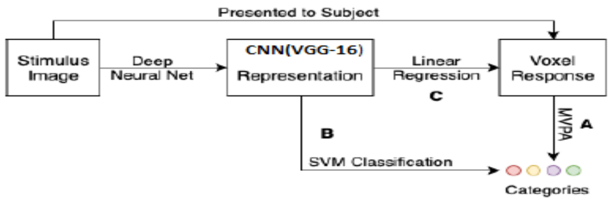
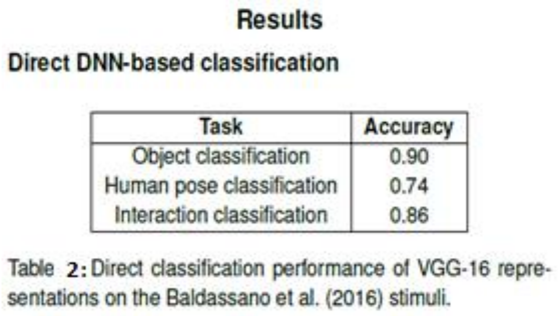


Fig. 15. Shows block diagram of the qualitative experiment

Four classes of images showing humans interacting with objects are taken like typing carrying objects etc. Test humans are asked to see the images and their brain functional Magnetic resonance Imaging(fMRI) is taken. The fMRI's Voxel (Volume equivalent of pixel) response are analyzed using Multivoxel pattern analysis (MVPA) which gives 4 distinct classes. This is the same experimental procedure done by Baldassano et al.

Same Images are fed to CNN and later classified by a SVM classifier as done by Aditi Jha et al. results obtained were Interesting. CNN is also able to classify into 4 categories like humans and much more as explained next.



Results

Direct DNN-based classification

Task	Accuracy
Object classification	0.90
Human pose classification	0.74
Interaction classification	0.86

Table 2: Direct classification performance of VGG-16 representations on the Baldassano et al. (2016) stimuli.

The set of CNNs trained on objects alone were able to predict objects well as in LOC part of visual cortex. Another set of CNNs that were trained on human poses was able to predict human poses as EBA in visual cortex. Also, CNNs trained on human-object interactions able to predict interactions as pSTS part in visual cortex. But CNNs trained to predict human object interactions failed to predict accurately the objects alone or human poses only. This was also the same case in pSTS part in visual cortex. So, this implies that the human object interaction is not a simple linear sum of human and object interaction but has nonlinear relation.

In many real-life applications such as object detection with different background scene, autonomous vehicles and industrial automation, we may have to detect humans or objects from the background. So, we need to inculcate linear compositionality in CNN. This can be done by a novel training method proposed in next section and an exhaustive experimental evaluation follows.

6) Algorithm for linear compositionality in CNN:

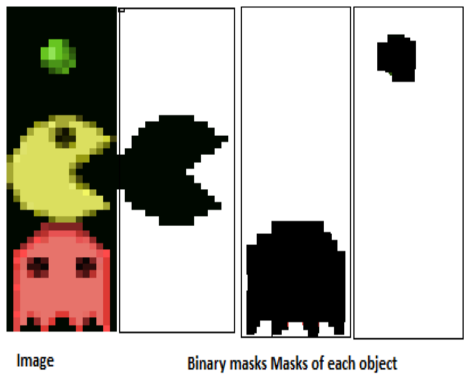


Fig. 16. Shows object on left most side and its binary binary mask m

Let X be an input Image with (x, y, c) dimensions. Let 'm' be a binary mask of an object in X.

m is a tensor with same shape of X with 1s indicating mask of an object. Let 'm' is of same size of X. Let $\emptyset$ be the 2D convolution activation function or a mapping between Image layer in a CNN. Let $\phi(X)$ is the 2D convolved output with a kernel having (h, w, c) dimension.

We define $\emptyset$ to be compositional if and only if

$$\emptyset(m \cdot X) = P(m) \cdot \emptyset(X) \quad \rightarrow (1)$$

where. Operator is an element-wise multiplication (not matrix multiplication).

Let p be a projection operator.

The projection p(m) down samples object mask m (x,y,c) to the size of output of $\phi(X)$ ie (h,w,c)

For Example: if $\phi(X)$ is the activations of size (h,w,c); where h,w are spatial dimensions and c is the feature (RGB-color)channels p(m) will downsample the object-mask m to size (h, w) and then stack c copies of the down-sampled object mask on top of each other to produce a mask of size (h, w, c).

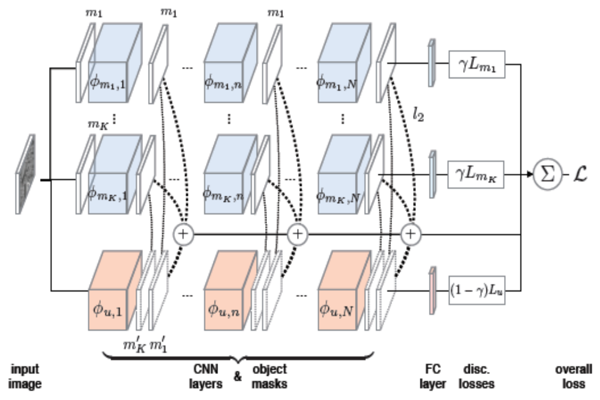


Fig. 17. Linearized CNN architecture for K objects

This architecture follows Equation (1). If input Image X has k objects then as per Equation (1) we will have k masked CNN and 1 unmasked CNNs .The (k+1)th CNN receives unmasked input.

The training method tries to reduce 2 types of losses.

Discriminative Loss (Ld.): For correct discrimination of objects we add this for all K masked and 1 unmasked CNN.

$$\mathcal{L}_d = \frac{1}{K}\left(\sum_k \gamma L_{m_k}\right) + (1-\gamma)L_u. \quad \text{-->(12)}$$

Where, L_{mk} is discriminative loss for K masked CNNs, *Lu* is the discriminative loss for 1 unmasked CNN and $\gamma \epsilon \{0,1\}$ is the Hyperparameter of CNN.

Compositional Loss (Lc): It is a l2 difference on activation of masked CNN and Activation of the unmasked function

$$\mathcal{L}_c = \frac{1}{K}\sum_k \sum_n \lambda_n \|\phi_{m_k,n} - \phi_{u,n} m'_k\|_2^2. \quad \text{--> (13)}$$

Where m'_k is a tensor of 1s for all k Objects or 0 otherwise.

λ_n is the layer specific penalty hyperparameter.

Final objective is L = Lc+Ld. We must reduce this L loss during the training.

7) *Simple experimental verification with single object and mask:*

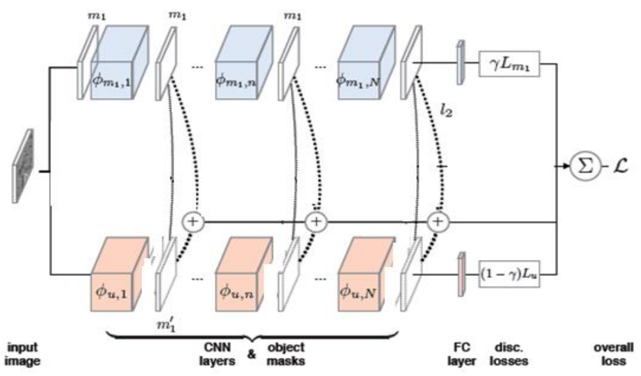


Fig. 18. Linearized CNN architecture for 1 object (K=1)

In this simplified experiment to inculcate linearity, we implement K =1, So we have 2 CNNs. We select 1 object from scene (accounts for masked CNN). We have 1 unmasked CNN which sees the entire scene. The Parameter (input features) space is shared by the 2 CNNs. We select the hyperparameter γ = 5. This model is 50% slower to train than a single CNN as there are 2 compositional CNNs.

To Minimize the loss function L, where L = lc + ld, the following method is used. Ld. is composed of L_{mk} and Lu are instantiated as softmax function and cross entropy is used to minimize it. Lc. is of standard Mean squared error (MSE) form(convex) and Stochastic Gradient Descent (SGD) can be used to minimize.

8) *Experimental verification:*

Experiments are performed on 3 types of image types.

a) *3D single and 3D multi images:*

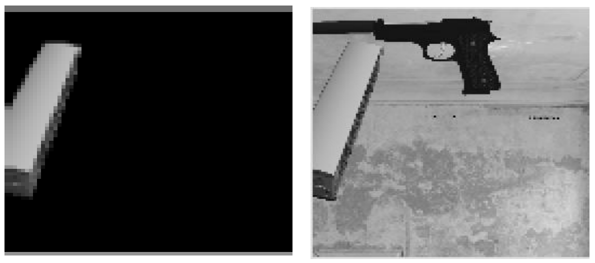


Fig. 19. Shows the test images of 3D single and 3D multi objects

Twelve -3D object classes (example: Guns, Bus etc.) containing 20 instance per class with 50 different perspective and with 20 different backgrounds. 3D-single **(**1600 images) shows single object in front of random backgrounds.3D-Multi (800 images) shows multi objects of various degrees of occlusion. We distinguish category level setting and between

3D-single and 3D-Multi images. There is an 80% training and 20% test set division among the images.

b) MNIST dataset:

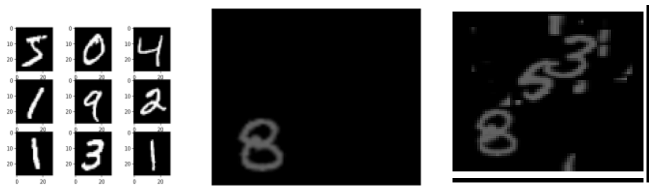


Fig. 20. Shows the MNIST images

These are images of handwriting of numbers. We divide this dataset as 3D-single and 3D-multi as per previous general rules.

c) MS-COCO dataset:

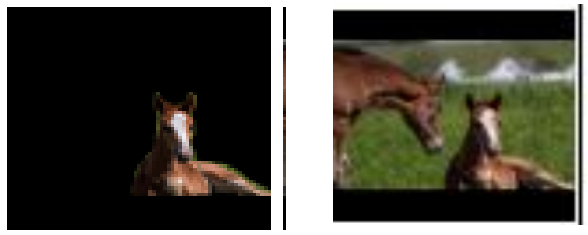


Fig. 21. Shows the MS-COCO images

These are images of objects in their natural contexts.1st test - we test classification performance on size of objects. 2nd Test – we test the classification for objects in and out of contexts.

9) Test Metrics:

For 3D Single,3D Multi and MNIST**,** performance is the average fraction of correctly predicted object classes among top-k scoring network predictions.

Where, k is the number of objects in the image.

For MS-COCO dataset: we treat object classes separately and report average precision (AP**).**

For all cases performance is monitored on a test over different epochs as training progresses.

10) Test comparison strategy:

Below are the versions of CNN versions tested.

COMP-FULL: Full compositional model.

COMP-OBJ-ONLY: COMP-FULL model but with only mask equal to 1s for k objects, no mask for background activation.

COMP-NO_MASK: COMP-FULL with no masks.

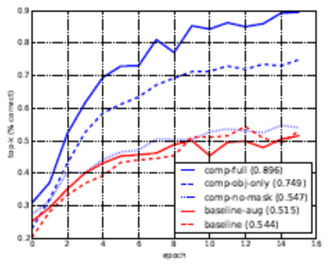
BASELINE: A standard CNN.

BASELINE-AUG: it splits each image into 2 equal parts and 1 part is put in black background and other half is put in cluttered background for each batch.

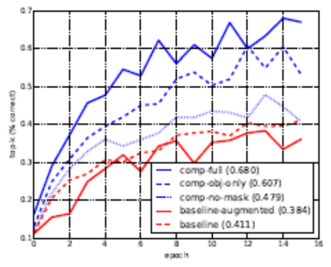
BASELINE-REG**:** BASELINE with regularization.

BASELINE-AUG-REG: BASELINE-AUG with regularization.

11) Results:

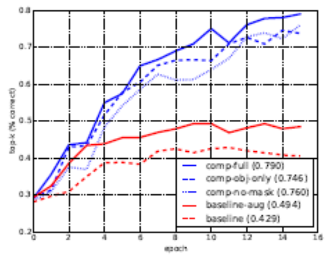


(a) 3D-Single-Inst

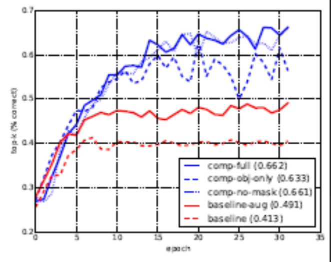


(b) 3D-Single

X- axis is epoch and y-axis are percentage accuracy of top-k. All variants of Compositional CNN perform well (Blue curves) esp. FULL_COMP. All baselines models show less accuracy esp. Baseline CNN.

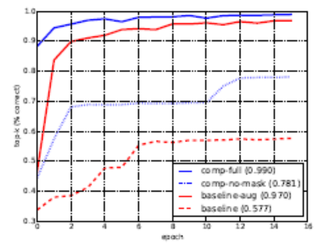


(c) 3D-Multi-Inst

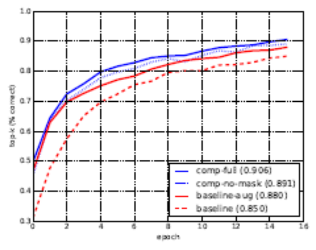


(d) 3D-Multi

For MNIST (handwritten Number images): The accuracy is almost same for compositional vs baseline, with COMP-FULL still beats baseline with small margin

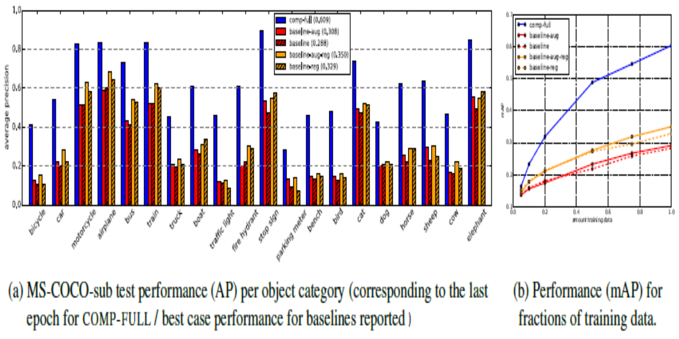


(e) MNIST-Single



(f) MNIST-Multi

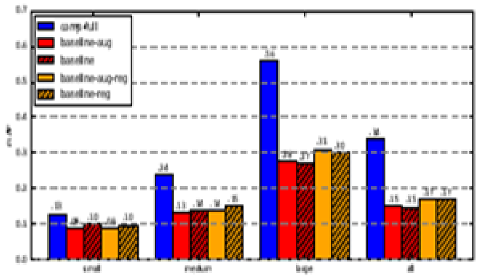
MS-COCO (objects in their natural settings) : The accuracy for COMP-FULL is higher by 25.5% than baseline CNN.



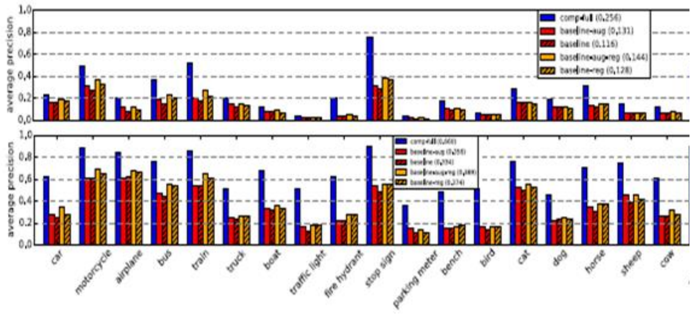
(a) MS-COCO-sub test performance (AP) per object category (corresponding to the last epoch for COMP-FULL / best case performance for baselines reported) (b) Performance (mAP) for fractions of training data.

(a)For MS-COCO dataset, the FULL composition model's performance exceeds by 32% for various image types

(b)Gives Average precision for different sizes of training data –Here also the COMP-FULL exceeds the others.

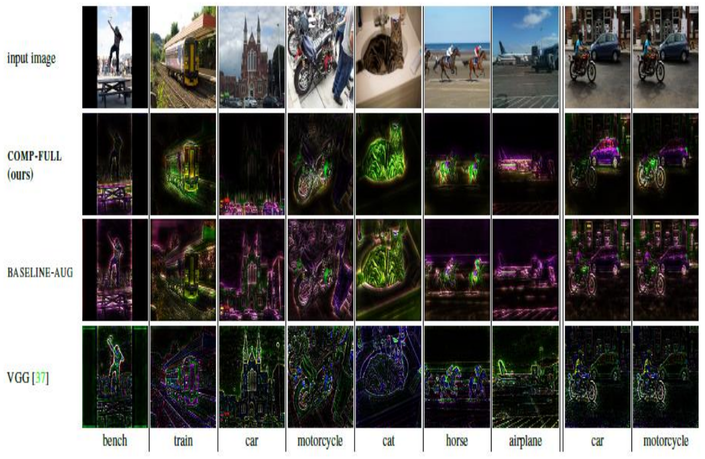


(c) MS-COCO-sub performance for objects of different sizes



(d) MS-COCO-sub performance for objects in-context (bottom) and out-of-context (top).

COMP-FULL performance exceeds by 25% over others for MS-COCO-sub.



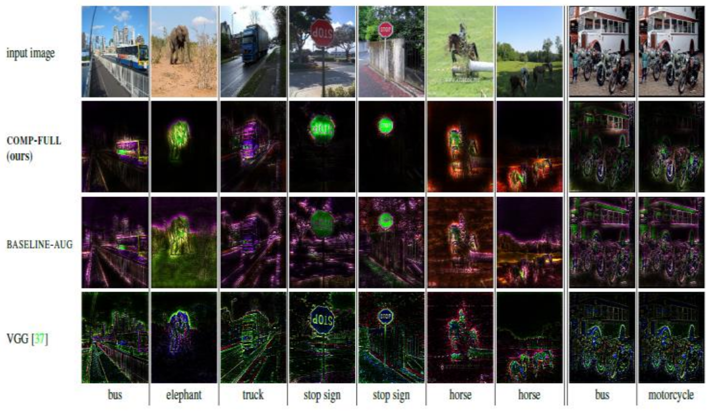


Fig. 22. Shows the MS-COCO dataset and various other CNN algorithms processed output images for qualitative analysis.

By the qualitative analysis of the images, it shows that COMP-FULL algorithm can detect the object from its surroundings well compared to baseline CNN and other algorithms.

IV. Conclusion:

The Subject of Compositionality in Modular Deep Neural networks inspired from biological brain were explored in depth using 2 instances. PART-A was modularization using EM algorithm which dynamically selects the best performing module for an input batch and assigns them thereby automatically selecting the best size of the deep neural network. This solves computation complexity by splitting big problem into smaller problem and time to train which are the 2 most important problem in deep networks. This also solves the problem of manually selecting the size (Layers) of DNN for a given task which can be inefficient and helps in transfer learning where pre trained modules can be mixed with new modules to reduce the training time.

In PART-B, we explored how the topic of computer vision takes inspiration from the biological visual cortex of the brain to do both linear and non-linear compositionality using CNN. We saw the baseline CNN exhibits non-linear compositionality and is not able to detect an object from its surroundings and required linear compositionality. We saw an algorithm that inculcated linearity by a modified training method, thereby making it detect an object from its surroundings more efficiently.

Going forward further study can be done to find the effects of EM Algorithm for large scale problems like RESNET and for small size datasets where It shows overfitting. Also the linearized CNN computation complexity is high as it employs k CNNs

References

[1] 7508 Modular networks learning to decompose neural computation – Louis Kirsch et al.

[2] Teaching compositionality to CNNs – Austin Stone et al.

[3] Do Deep Neural networks model Nonlinear compositionality in neural representations of human object interactions – Aditi Jha et al.

[4] https://machinelearningmastery.com/expectation-maximization-em-algorithm/

[5] Deep learning by Andrew Ng – Coursera

[6] http://blog.ivank.net/viterbi-algorithm-clarified.html

[7] Wikipedia

[8] https://en.wikipedia.org/wiki/Convolutional_neural_network

[9] https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

[10] 2015 Springer Handbook of computations