GuideBP: Guided Backpropagation in multi-output neural networks by channeling gradients through weaker logits

Swarnendu Ghosh[1,*], Bodhisatwa Mandal[2], Teresa Gonçalves[3], Paulo Quaresma[3], Mita Nasipuri[2], and Nibaran Das[2]

[1] Institute of Engineering & Management, Kolkata 700091, WB, India
University of Engineering & Management, Kolkata 700091, WB, India
IEM Centre of Excellence for Data Science, Kolkata 700091, WB, India
* Corresponding Author: drghosh90@gmail.com
[2] Jadavpur University, Kolkata 700032, WB, India
bodhisatwam@gmail.com, nibaran.das@jadavpuruniversity.in,
mita.nasipuri@jadavpuruniversity.in
[3] University of Évora, Évora 7004-516,Portugal
tcg@uevora.pt, pq@uevora.pt

Abstract. Convolutional neural networks often generate multiple logits from multiple networks. In most cases we use simple techniques like addition or column averaging for loss computation. But this allows gradients to be distributed equally among all paths. The proposed approach attempts to guide the gradients of backpropagation along weakest branches of the neural network. A weakness score is proposed that defines the class specific performance of individual logits. This is then used to create a new output distribution that would guide gradients along the weakest pathways. The proposed approach has been shown to perform better than traditional column merging techniques and can be used in several application scenarios. Not only can the proposed model be used as an efficient technique for training multiple instances of a model parallelly, but also CNNs with multiple output branches have been shown to perform better with the proposed upgrade. Various experiments establish the flexibility of the learning technique which is simple yet effective in various multi-objective scenarios both empirically and statistically.

Keywords: Backpropagation, Neural Networks, Deep Learning, Convolutional Neural Networks, Ensemble Methods

1 Introduction

Convolutional Neural Networks(CNNs) have come a long way since their introduction of the LeNet5 in 1998[7]. Over the last two decades, several architectures have been proposed to solve different types of computer vision tasks[8, 3]. Throughout the evolution of CNNs, it has been observed that the topology in which the layers of CNNs are arranged has a significant impact on performance[4]. In the earlier years alternating convolution and pooling layers were

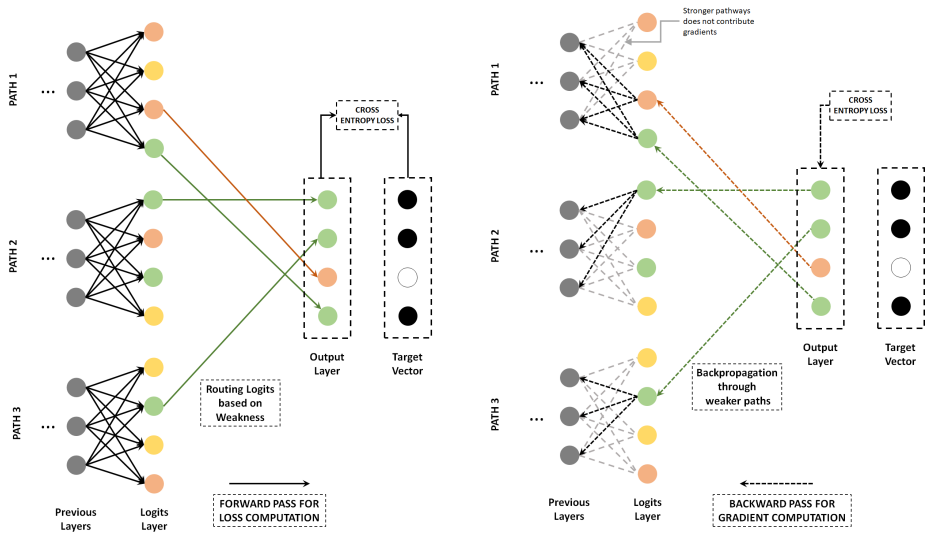

Fig. 1. Guiding backpropagation through weaker pathways: The black and white dots signify 0 and 1 respectively in the target vector. The green-yellow-red dots signify output neurons. Green signifying high values, red signifying low values and yellow is something in the middle.

used for feature extraction before passing them to a fully connected layers for mapping the features to the output [7, 6, 9]. Deeper networks were proposed in subsequent years to improve upon previous models. Convolutional blocks with multiple convolution and batch normalization layers were used for extracting rich features at different scales [13]. With deeper networks, the flow of gradients through long chains of computation became a challenge due to issues like vanishing gradients. In this context, major strides have been made by manipulating the CNN architecture. GoogLeNet[14] used auxiliary classifiers to improve gradients of intermediate layers while ResNet and DenseNet implemented skip connections for better gradient flow through short-cut channels. While standard networks like these are good at learning a set of generalized features, multi-column architectures aim for a much more distributed learning protocol[1, 2] where each column is designed to attend a specific purpose. Multiple columns drawn from different layers of a CNN can be used for mapping features of different scales to the output space. This is commonly used of object detection [10, 11] where objects of interest can vary in terms of scale in the image. For datasets like handwritten texts where samples adhere to a geometric template, region specific columns have been quite successful [12, 15, 16]. However, in most multi column architectures, the columns are either trained independently or parallelly. While each column specializes in learning specific features, their contribution to the output distribution has no effect on the training process. The most common

approach to deal with multi path network involves individual training of each path using separate loss functions or adding or averaging the logits.

In the proposed methodology a novel learning technique has been developed which emphasizes on exclusively improving the pathways of multi column networks that contribute adversely to output layer. Unlike other aggregation methods like adding or averaging logits, in this method the gradient isn't equally split along all pathways. The learning model considers multiple output distributions from individual columns and forwards the most uncertain pathways to the prediction layer for guiding the backpropagation algorithm along the weakly learned features. This allows all columns of the networks to perform at par with each other. The proposed learning technique can be used wherever a CNN has multiple branches connecting to the output layer. Various application scenarios have been demonstrated that employs the proposed technique to avoid a greedy approach for column optimizations.

Objective The objective of the proposed work is to provide a more efficient way to combine multiple sets of logits generated by multi path convolutional neural networks. More specifically, the major contribution lies in the method of combining the logits and establishing as a superior method over standard techniques like adding or averaging. Since we are focusing on a generic learning mechanism for multi-path convolutional neural networks, the proposed method is not being established as a bench-marking model. As most of the applications demonstrated in the work are built as an upgrade over existing models[5, 12, 14, 11], it comes with some of the drawbacks of the back-end models themselves.

2 Multi-objective optimizations in CNNs

Convolutional neural networks are traditionally trained by computing gradients corresponding to trainable parameters that contribute towards a loss function. A stochastic gradient descent based loss minimization technique gradually pushes the weights towards minima in the loss versus weight space. The loss function has significant impact on several factors that control the performance of the network. While stochastic approaches like gradient descent makes the network prone to issues like local optima and over-fitting, loss functions involving multiple output vectors can negate these issues to a considerable extent as compared to loss functions corresponding to a single output vector. While coping up with multiple objectives a generalized learning paradigm is enforced. In practice, multiple objectives can be enforced at various levels. While some approaches tune differently initialized instances of the same model, other approaches consider things features of multiple scales from different layers, different regions of the feature space and even different regions of the input space.

2.1 Ensemble of multiple instances of a CNN

One of the most coarse approach for multi-objective is to take an ensemble of fully functional CNNs. CNNs, by their design, depends on the initialization of

the parameters. By taking multiple instances of the same model, we can explore weight space starting from different initial positions. Such models can be either trained individually and combined by ensemble techniques such as max voting, or have their logits summed or averaged before computation of the loss function. Either way, the individual pathways are trained greedily and do not approach to attain the objective jointly.

2.2 Using auxiliary classifiers to facilitate gradient propagation

With the expansion of network depth, the chance of gradients vanishing in earlier layers is increasing due to the use of successive products partial derivatives. In networks like GoogLeNet, and Inception Net, the gradients in the earlier layers are boosted using a set of auxiliary classifiers drawn from features of intermediate layers. The purpose of these layers is to introduce an additive component to the gradients of the earlier layers to make up for the drop in the value of the gradients due the depth of the network. Though auxiliary classifiers were optimized according to the same ground-truth as the output layer, however, their learning strategy is greedy. While the gradients in the common pathways were summed, there is no control of the pathways with regards to the strength of the performance. For example, a component of the logits from the auxiliary classifier corresponding to a specific class may be better at contributing to the performance as compared to their equivalent counterpart in the output layer. This can happen due to the fact that all classes do not require the same amount of depth to model the concepts. Excess amount of depth in the network can overfit the training data corresponding to samples of some class.

2.3 Multi scale region extraction from feature space

Object detection in the real world is a particularly tricky task because of the variable shape and size of objects in the scene. Popular object detection networks like YOLO thus highly depends on extracting features from multiple scales, to create a scale space pyramid of anchor boxes that can serve as candidates for bounding boxes. The YOLO model divides the feature space of the intermediate layers into grids of regions which are then used for detecting the presence of a bounding box centre. However, these multiple regions only contribute to recommendation of separate candidate boxes. In the proposed work we explore a variation different regions of the feature space has been shown to contribute to a unified learning process while solving a common objective.

2.4 Multi-column networks operating on different regions of the input

For datasets of small size and low intra-class variance and inter class variance, it is often seen that operating with specific regions of the input yields better results because different regions of the the sample contributes differently towards the

modeling of the output space. The input itself can be divided into different sizes of grids where multi-column convolutional neural networks can work on each of these grid individually. Such multi-column networks are either trained individually or their logits are summed or averaged to compute loss against the ground truth. Either way, the weaknesses of each branch are untracked and thus leaves room for improvement.

3 Guiding backpropagation through weaker pathways

The purpose of the current work is to increase the effect of backpropagation on the more uncertain pathways in a multi-path network. The goal is to stop the pathways that have learnt strong class specific concepts from contributing to the gradient. The weights of the network should give more preference on repairing the weaker concepts modeled in the network. This desired effect of the proposed work is visualized in fig. 1.

The method requires a set of N number of class specific logits, which corresponds to N different pathways. A final set of logits is computed by choosing the most uncertain class specific components across each pathway and combining them. The computation involves calculating the weakness of each component of the predictions given by each path. Finally, a new logit is formed by routing the weakest components among each path. An illustration of this is shown in 2. The method is discussed in details in the subsections below.

3.1 Computing weakness of predictions

The process requires a N number of C dimensional logits, N is the number of **outputs** and C is the number of **classes**. Ideally, the logits(denoted as $\mathbf{u}$) of the last layer of a path are supposed to reflect the expectation of the image to belong to a specific class. Since we are dealing with multiple pathways, the logits are passed through a log-softmax function before proceeding to obtain a distribution $\hat{\mathbf{u}}$.

$$\hat{\mathbf{u}}_{ij} = \log \frac{\exp(\mathbf{u}_{ij})}{\sum_{k=1}^{C} \exp(\mathbf{u}_{kj})}, \tag{1}$$

where, $i \in \{1, C\}$ and $j \in \{1, N\}$. The log-softmax is calculated to obtain log probabilities of the logits. It is preferred over standard normalization or whitening transforms to enhance maximum expectations and penalize misclassifications severely.

The weakness of a prediction is represented by the relative value of $\hat{\mathbf{u}}$ among all the pathways. For a target one-hot vector $\mathbf{t}$, the weakness of the log-softmaxed logits can be defined by a function (W) such that

$$W(\hat{\mathbf{u}}_{ij}) = \mathbf{t_i} + (-1)^{(1-\mathbf{t_i})} \times \frac{\hat{\mathbf{u}}_{ij}}{\sum_{m=1}^{N} \hat{\mathbf{u}}_{im}}, \tag{2}$$

where, $i \in \{1, C\}$ and $j \in \{1, N\}$. The weakness value maximizes the pathway with the highest logits corresponding to the negative classes and lowest logits

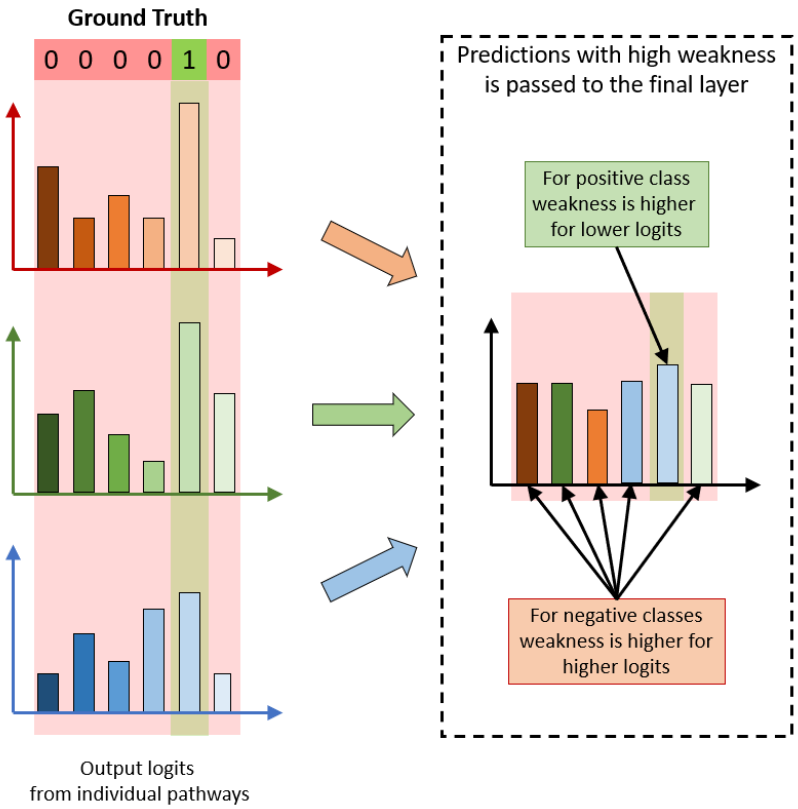

Fig. 2. Generating combined logits by routing the weakest components corresponding to each class

corresponding to the positive class. The intuition behind the idea is that the strength of a classifier depends on how high valued logits it produces for the positive class and vice versa. Hence, the weakness is defined as the opposite of that. These weakness values corresponding to each of the log-softmaxed logits are used to build the combined logits for channeling the backpropagation algorithm.

3.2 Building logits with most uncertain pathways

While building the combined logits the focus should be on choosing the pathway with the highest weakness associated with it for the backpropagation algorithm to tune. The combined logits $\mathbf{v}$ is defined as,

$$\mathbf{v}_i = \arg\max_j \; W(\hat{\mathbf{u}}_{ij}) \tag{3}$$

The utility of composing logits with the weakest components of each pathway, is guiding the backpropagation through the weakest branches first. Throughout the

learning phase, the backpropagation will always channelized along the pathway that is producing the most uncertain response among all the different branches, thus improving the global certainty of all the branches.

3.3 Inference

Once the multi-path network has been trained, the inference can be carried out using one of the two methods mentioned below.

Combining strongest components of each pathway During evaluation of the network, the least weakest components are selected from each branch. Since the target vector $\mathbf{t}$ is not available during evaluation, a pseudo target vector $\hat{\mathbf{t}}$ is composed as,

$$\hat{\mathbf{t}}_i = \begin{cases} 1, & \text{if } \hat{\mathbf{u}}_{ij} = \max_j(\hat{\mathbf{u}}_{ij}) \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where, $i \in \{1, C\}$ and $j \in \{1, N\}$. The weakness function is now computed with respect to the pseudo target vector $\hat{\mathbf{t}}$ as

$$W(\hat{\mathbf{u}}_{ij}) = \hat{\mathbf{t}}_i + (-1)^{(1-\hat{\mathbf{t}}_i)} \times \frac{\hat{\mathbf{u}}_{ij}}{\sum_{m=1}^{N} \hat{\mathbf{u}}_{im}}, \tag{5}$$

where, $i \in \{1, C\}$ and $j \in \{1, N\}$. The final logits $\hat{\mathbf{v}}$ is calculated with respect to pathways with minimum weakness value as follows:

$$\hat{\mathbf{v}}_i = \arg\min_j \; W(\hat{\mathbf{u}}_{ij}) \tag{6}$$

where, $i \in \{1, C\}$ and $j \in \{1, N\}$.

Averaging the logits from each pathway Multi-path networks can be trained by adding or averaging logits from each path and back propagating with equal preference. However, in the proposed method, the network focuses on improving the weakest components of each logit vector in every iteration. Hence, at the end of the training, the proposed method provides stronger individual branches as compared to the traditional method of adding or averaging logits. After training, if a mean of logits is considered for inference purpose a significant boost can be seen as the individual pathways have lesser weaknesses. In this way the resultant logits for inference may be written as

$$\hat{\mathbf{v}}_i = \frac{1}{N} \sum_j \hat{\mathbf{u}}_{ij} \tag{7}$$

where, $i \in \{1, C\}$ and $j \in \{1, N\}$.

3.4 Understanding the intuition

In standard column merging techniques like averaging or individual training, there is no constraint on the columns to recognize the strengths of the concepts learned by them. While averaging multiple columns often improve performance, it depends on the performance individual columns as well. It is often said that a team is as strong as its weakest player. Hence if one of the pathways is weaker than compared to the others, the effect of averaging would be diminished. The proposed methodology aims to address this. At any given iterations, the wrongest predictions would be attended to at first. This ensures all the classifier to perform at a similar level. This, in turn, improves the effect of column merging.

Fig. 3 demonstrates a simulation of how the proposed approach can affect the logits in successive iterations. The values do not belong to an actual experiment but appropriately set for a better understanding of the intuition. The green arrows represent logits corresponding to the positive class. Here the value farthest from 1.0 is attended at first while the others do not affect the gradients. The red arrows corresponding to the negative class attend to the values farthest from 0.0.

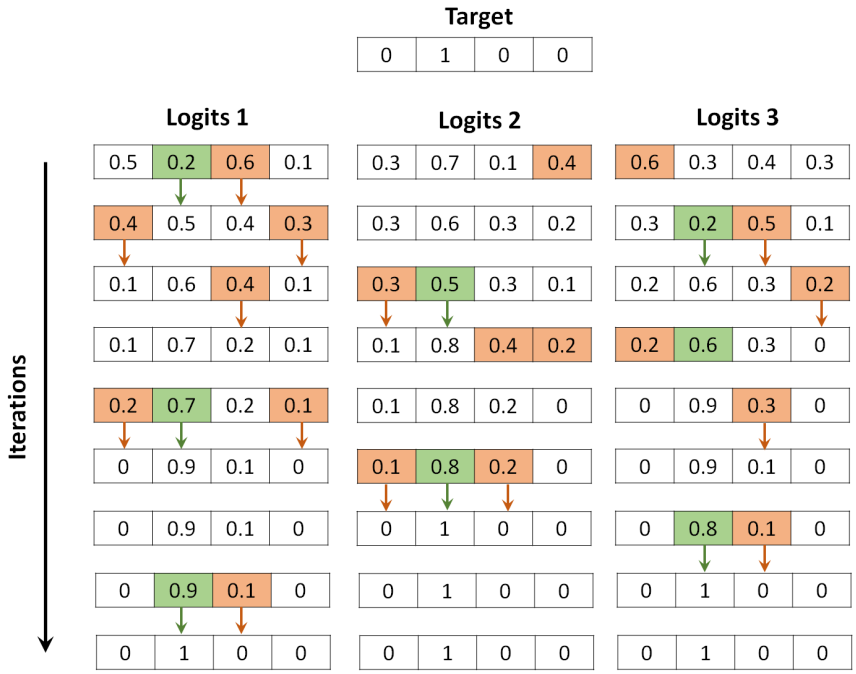

Fig. 3. A simulation of logits progressing through iterations. Green arrows signify the positive class and red arrows signify negative classes. The colored cells when combined creates the final logits through which backpropagation occurs.

4 Applications

The proposed methodology can be applied in multiple scenarios wherever we need to combine logits coming from different path. Till now the available options for dealing with logits was to either train them individually and use classifier ensemble techniques or compute the sum or mean of the logits to calculate the training loss. In the current approach, we have demonstrated four use cases where the proposed method can be used. The four use cases cover the various level of depths and locality of features.

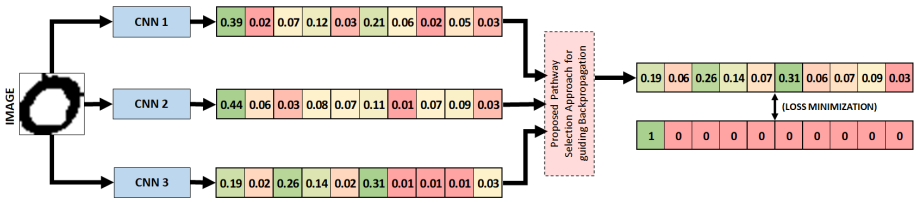

Fig. 4. Model **M1**: Logits drawn from multiple instances of a model. ResNet18s[5] were used in the experiments as the backend CNN

4.1 Logits from multiple instances of a model (M1)

One of the most straight forward implementations of the proposed methodology is to use the method as an ensemble technique. Here we take log-softmax of the logits from individual CNN based classifier as input. The proposed approach provides us with a modified logits composed of the weakest components from each classifier. The backpropagation thus forces each of the CNNs to update its weight to enhance their weakest concepts. Fig. 4 demonstrates an example where outputs from three classifiers are combined using the proposed approach. Unlike traditional ensemble techniques, it is not necessary to train the networks individually. Simply averaging logits does not force the entire network to train explicitly with respect to the weaker components. In our experiments, we have combined logits from three 18 layer resnet models.

4.2 Logits from different layers of a model (M2)

For very deep networks auxiliary classifiers often serve as gradient boosters for earlier layers. In the GoogLeNet model, two auxiliary classifiers are implemented during training in addition to the final classifier. The auxiliary classifiers are drawn from the end of the third and fifth inception blocks. However, these classifiers serve no purpose during the inference phase is hence chopped from the network once the training completes. These logits are drawn from layers with different levels of complexity of features. Obviously deeper networks will produce

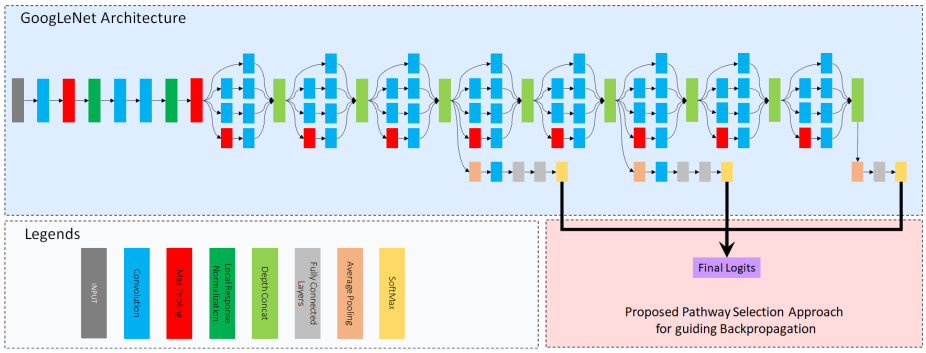

Fig. 5. Model **M2**: Logits drawn from features with different scales. Based on a GoogLeNet[14] backend.

much smaller feature maps that model global concepts as compared to shallower features which capture granular features in larger feature maps. The proposed approach aims to bank on this factor to capture both shallow and deep features and train the preceding weights exclusively with respect to weaker components of the predicted logits. Using the proposed approach, the final logits and the two auxiliary logits are combined during training. Unlike the original network, the auxiliary classifiers contribute during the inference as well as mentioned in section 3.3.

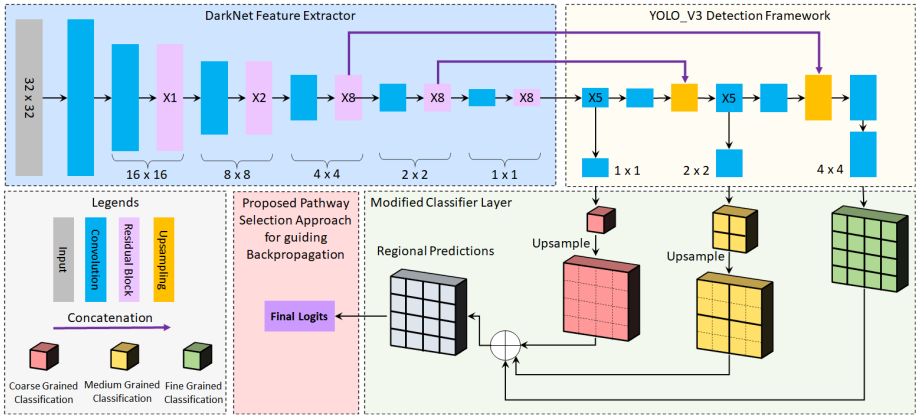

Fig. 6. Model **M3**: Logits drawn from sub-regions in the feature space. Based on a DarkNet and YOLO_v3 [10] backend

Table 1. Dataset Description

Dataset Name	Description	Classes	Channels	Train Set	Test Set
CMATERdb 3.1.1	Bangla handwritten digits	10	1	4000	2000
CMATERdb 3.2.1	Devanagari handwritten digits	10	1	2000	1000
CMATERdb 3.3.1	Arabic handwritten digits	10	1	2000	1000
CMATERdb 3.4.1	Telugu handwritten digits	10	1	2000	1000
CMATERdb 3.1.2	Bangla handwritten basic-characters	50	1	12000	3000
CMATERdb 3.1.3	Bangla handwritten compound-characters	199	1	34229	8468
MNIST	English handwritten digits	10	1	60000	10000
Fashion-MNIST	MNIST like dataset of fashion products	10	1	60000	10000
notMNIST	MNIST like dataset of A to J in various typefaces	10	1	60000	10000
Kannada-MNIST	MNIST like dataset of Kannada handwritten digits	10	1	60000	10000
Kuzushiji-MNIST	MNIST like dataset of Japanese handwritten characters	10	1	60000	10000
SVHN	English digit dataset from street view images	10	3	73257	26032
Cifar10	Images of objects occurring in natural scenes	10	3	50000	10000

4.3 Logits from feature subspace (M3)

Objects can appear in different sizes and position throughout the space of the image and this makes it particularly hard for object detection approaches to locate the object in the scene. This challenge also exists for object recognition tasks. Different size and position of objects can be activated not only features at different depths but also at different positions. Networks like YOLO_V3 divides features of various depths into grids of different sizes which are then used to locate centres of bounding boxes. Our proposition modifies the classification layer of the network. With an input of 32×32 the network produces three regional predictions of size 1×1, 2×2, and 4×4, which correspond to the entire feature map, each quadrant and each decahexadrants of the feature map respectively. Buy up-scaling the 1×1 and 2×2 feature maps to 4×4 using nearest neighbor interpolation and adding them we can obtain 16 predictions corresponding to 16 different regions of the feature space. The proposed approach can then combine the 16 prediction vectors corresponding to the 4×4 grid of the feature space to form the final logits. In this case, the model attempts to train different part of the networks to attend to the weakness of different regional concepts with respect to the different classes.

4.4 Logits from input subspace(M4)

Just like the YOLO based network computed the logits from sub-regions in the feature space, it is also possible to divide the inputs to different regions and run parallel multi column deep neural networks on individual regions. The columns can either be combined by averaging the softmax of the logits or by averaging them. In both cases the individual networks set no preference on the strength of the concepts learned by them. The network proposed in the works of [12] combines 23 CNNs working on regions of different sizes corresponding to different positions. The proposed approach combines those logits to create a model where all columns can be trained together and they focus on strengthening their weakest concepts.

Table 2. Results of all experiments

Datasets	Logits from multiple instances of a model (M1)				Logits from different layers of a model (M2)				Logits from feature subspaces (M3)			Logits from input subspaces (M4)		
	Single ResNet18	Average of 3 Logits	Strong Inference	Mean Inference	Native GoogLeNet	Average of 3 Logits	Strong Inference	Mean Inference	Average of 16 grids	Strong Inference	Mean Inference	Average of 23 Branches	Strong Inference	Mean Inference
CMATERdb 3.1.1	99.20	99.50	99.70	**99.80**	96.70	98.70	99.00	99.30	98.70	99.70	**99.80**	99.10	99.40	99.50
CMATERdb 3.2.1	99.30	99.48	99.55	99.50	97.50	99.35	99.35	99.50	99.40	99.60	**99.85**	99.10	99.30	99.60
CMATERdb 3.3.1	99.40	99.48	99.70	99.70	98.40	98.90	99.30	99.60	99.30	99.70	**99.90**	98.70	99.10	99.40
CMATERdb 3.4.1	98.90	99.10	99.30	99.50	97.20	98.70	99.20	99.60	99.00	**99.80**	**99.80**	98.90	99.20	99.70
CMATERdb 3.1.2	97.67	97.80	98.40	98.40	98.00	98.23	98.23	98.33	98.50	98.63	**98.90**	97.67	97.80	98.00
CMATERdb 3.1.3	96.40	96.70	**97.90**	97.70	96.90	97.32	97.33	97.67	97.20	**97.90**	97.70	94.70	96.32	96.50
MNIST	99.63	99.63	99.70	**99.73**	99.58	99.59	99.73	**99.73**	99.55	99.65	99.65	99.29	99.25	99.40
Fashion MNIST	94.00	94.28	94.40	94.50	94.30	94.45	94.53	94.50	92.26	94.01	**94.60**	93.09	92.40	94.00
notMNIST	96.80	97.00	97.71	**97.73**	97.12	97.42	97.71	**97.73**	97.02	97.09	97.33	95.98	96.02	96.67
Kannada-MNIST	97.40	97.63	**98.87**	98.40	96.67	97.02	**98.87**	98.60	96.67	98.85	98.50	96.40	98.54	98.50
Kuzushiji-MNIST	96.67	97.20	97.87	98.00	97.40	97.66	**98.10**	**98.10**	97.20	97.61	97.67	96.00	96.77	97.20
Svhn	95.50	95.67	96.00	96.00	95.82	95.85	96.12	**96.20**	84.19	85.94	86.40	89.47	89.86	91.33
Cifar10	94.19	94.59	94.82	**95.20**	91.94	92.02	92.36	92.50	72.80	73.77	75.00	84.53	84.67	85.00
Mean	97.31	97.54	97.99	**98.01**	96.73	97.32	97.68	**97.80**	94.75	95.56	**95.78**	95.61	96.05	**96.52**

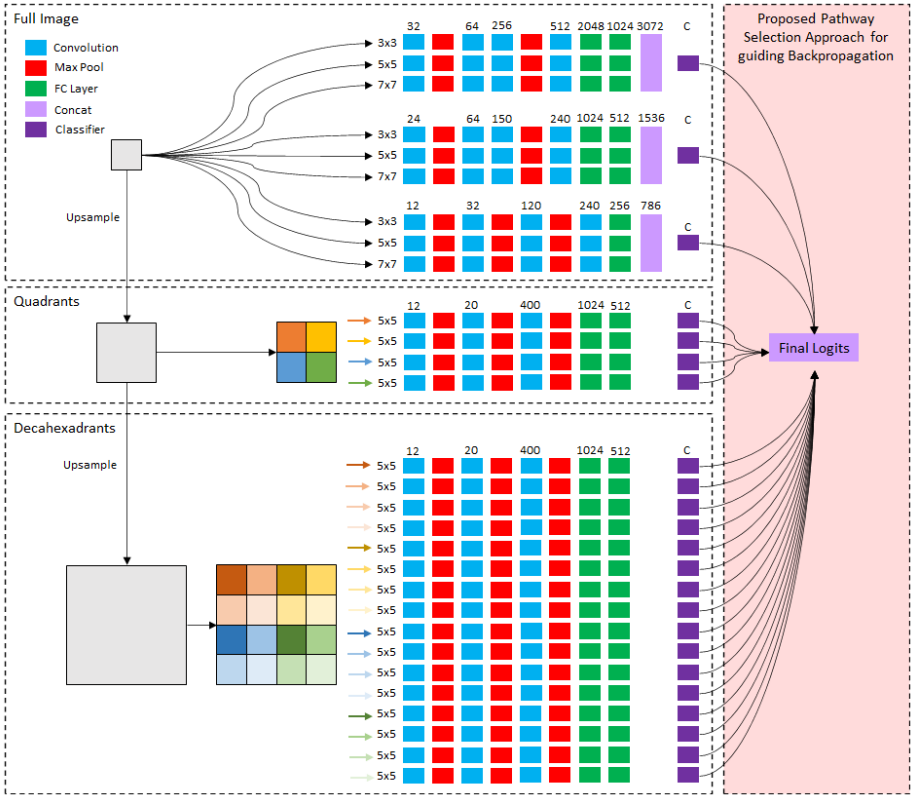

Fig. 7. Model **M4**: Logits drawn from sub-regions in the input space. Inspired from the Quad Tree based architecture[12]

5 Results and Discussions

Several experiments were carried out to validate the method of combination of multi column networks. The focus of the experiments was to prove that by training multi column models by the proposed method is more efficient than by averaging the outputs of the different columns. The inference was carried out by two different ways as mentioned section 3.3 and 3.3. The proposed method was tested in 4 different application scenarios as mentioned in section 4.1 - 4.4. The experiments were carried out on 14 different datasets that can be categorized based on various parameters.

5.1 Dataset Descriptions

The CMATERdb datasets consist of handwritten digits and characters from Indic scripts. There are 4 handwritten digit datasets with around a 200-400 hundred samples per class. Each of the digit datasets has samples divided into

10 classes corresponding to the digits. Additionally, there are 2 datasets focusing on characters. One of them has 50 classes and 240 samples per class and the other has 199 classes and around 172 samples per class. These datasets with higher number of classes and comparatively lower number of samples pose a serious challenge. We also look in datasets with the higher number of samples as well. Along with the MNIST dataset, there are 4 other datasets that follow the distribution pattern of MNIST. Each of them consists of 10 classes and around 6000 samples per class and focus on handwritten digits and characters from Kannada and Japanese, English alphabets in various typefaces and images related to fashion. All the previously mentioned datasets were in grayscale. Three RGB datasets were also considered. The SVHN dataset consists of digits occurring in natural scenes. It has 10 classes and average of 7325.7 training samples per class. Finally, the CIFAR 10 dataset was considered where natural scene objects are divided into 10 categories. No dataset augmentations were used but the inputs were normalized to a zero mean and unity variance. The dataset descriptions can be found in table 1.

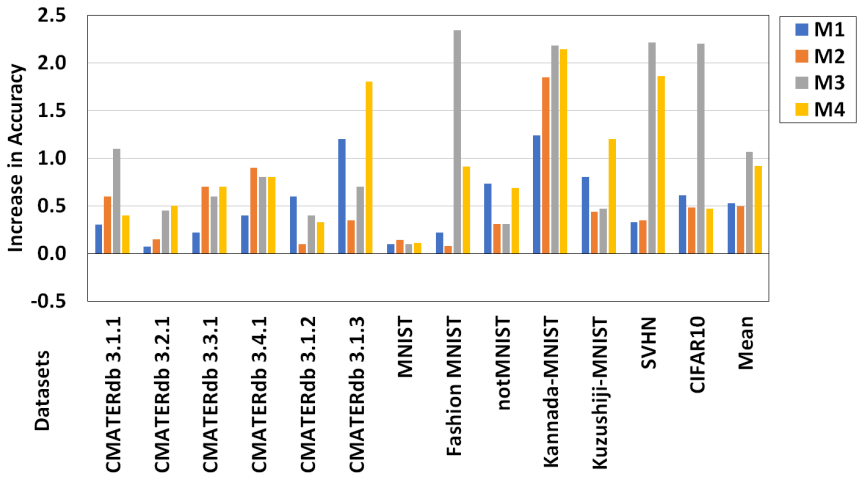

Fig. 8. Increase in Accuracy of the best of mean and strong inference protocol with respect to traditional averaging of logits

5.2 Experimental Results

As mentioned before 4 models have been proposed in section 4. The 4 models are built upon partially or fully adapted versions resnet18, googlenet, YOLO_v3 and the Quad Tree based MCDNN. No pre-trained models were used and all training was done from freshly initialized network. The networks were trained

and the model with the best training accuracy was chosen for testing. The use of multiple columns itself has a regularization effect on the training process hence early stopping was not necessary. Adabound optimizer was used to update the weights to reduce the cross entropy loss between the combined logits and the ground truth. In each of the four proposed models, we have two inference protocols as mentioned in section 3.3. "Strong Inference" corresponds to the method described in section 3.3 and "Mean Inference" corresponds to the method described in section 3.3. In all the 4 cases, we have compared against a version of the models where we have simply averaged the logits using in the traditional way during training. Additionally, a comparison has also been against the native model as it was proposed. The result of all the experiments is compiled in table 2.

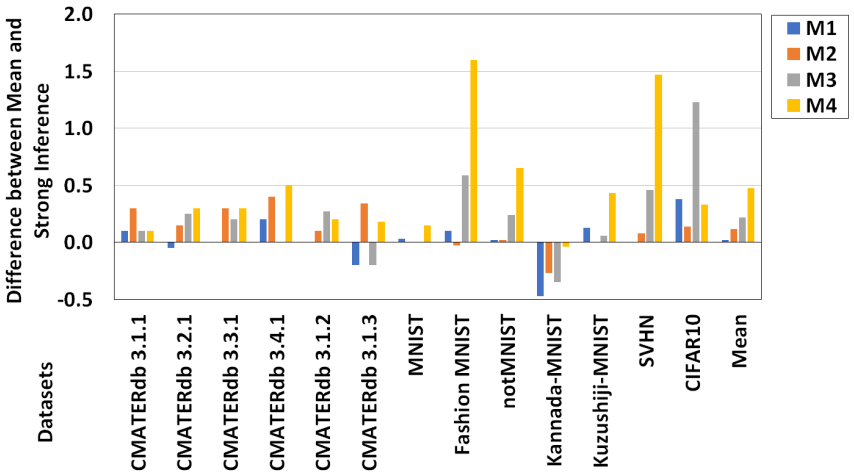

Fig. 9. Difference between the Mean Inference Protocol vs the Strong Inference Protocol (Mean-Strong)

5.3 Observations

If we observe the results in the table 2 we can draw some inferences with respect to the performance of the proposed models. The inferences are enumerated as follows:

1. Either of the proposed inference protocols always triumph over the traditional method of averaging multiple logits. The observation is demonstrated in fig. 8. The trend is also or native designs of the back-end network like standalone ResNet18 or GoogLeNet.

Table 3. McNemar Test between various proposed methods and existing networks to demonstrate significant difference among the classifiers ($p<0.05$ suggests that difference is significant)

Datasets	ResNet18 vs M1 Mean	ResNet18 vs M1 Strong	M1 Mean vs M1 Strong	GoogleLeNet vs M2 Mean	GoogleLeNet vs M2 Strong	M2 Strong vs M2 Mean	M3 Mean vs M3 Strong	M4 Mean vs M4 Strong
CMATERdb 3.1.1	0.2467	0.2344	0.1274	0.0126	0.0084	0.0973	0.0071	0.0017
CMATERdb 3.2.1	0.2253	0.1742	0.1364	0.0095	0.0077	0.1012	0.0003	0.0010
CMATERdb 3.3.1	0.1516	0.1056	0.0912	0.0110	0.0108	0.3113	0.0014	0.0011
CMATERdb 3.4.1	0.2833	0.2036	0.0508	0.0129	0.0115	0.0884	0.0012	0.0022
CMATERdb 3.1.2	0.1139	0.0451	0.0000	0.0154	0.0118	0.0146	0.0528	0.0088
CMATERdb 3.1.3	0.1087	0.0078	0.0000	0.0176	0.0091	0.0133	0.0096	0.0000
MNIST	0.4772	0.5987	0.3726	0.5630	0.3753	0.4510	0.1368	0.3147
Fashion-MNIST	0.1638	0.1065	0.0546	0.1464	0.0000	0.1724	0.0012	0.0000
NotMNIST	0.1716	0.1013	0.1142	0.1954	0.0001	0.0430	0.0437	0.0000
Kannada-MNIST	0.2009	0.0007	0.0062	0.1735	0.0320	0.0098	0.0000	0.0426
Kuzushiji-MNIST	0.1863	0.1139	0.2366	0.2573	0.0175	0.0226	0.0000	0.0387
SVHN	0.1039	0.0023	0.0040	0.0956	0.0001	0.0001	0.0000	0.0067
Cifar10	0.0064	0.0964	0.0736	0.1007	0.0411	0.0012	0.0141	0.0785

2. The ”mean inference” protocol performs best in several datasets, but the "strong inference" protocol has also shown better performance in some of them as well. This is shown in fig. 9.
3. If the average trend is followed in table 2, M1 produces the best results followed by M2, M4 and M3 successively. Several trends can also be noticed under careful observations.
   (a) M3 produces the best performance for all the 6 CMATERdb datasets with lower number of samples. This trend is also seen for one of the larger gray-scale dataset namely Fashion-MNIST. Small datasets that adhere to a basic template, benefits from the region specific learning technique. The effect of feature subspace analysis is also particularly beneficial for Fashion-MNIST where the samples correspond fashion attires that generally follow a relatively strong predefined template with low variance.
   (b) M2 produces the best results for 4 out of 5 MNIST like datasets and SVHN which is a RGB dataset. The intra-class variance for larger dataset is much higher hence feature subspace cannot be exploited as effectively. Fetching features from different layers can combine the effects of features of different sizes that are captured in layers of different depths of the network.
   (c) As CIFAR 10 consists of naturally occurring objects they exhibit a high degree of intra class spatial variance. In other words, the class specific template has a high variance. Hence, combining multiple instances of whole models performs the best. The concepts required for classification are too abstract to be extracted from shallower depths.

4. Finally from the McNemar test conducted between the strong and mean inference methods as well as their comparison with standard networks like ResNet18 or GoogLeNet we can draw some conclusions. The p-values of the McNemar test are provided in table 3. Among the four proposed model we can observe that other than M1, all the other models are showing significant difference in the learned concepts. Among the datasets only MNIST dataset didn't demonstrate statistical significance. This is probably due to the saturation in the performance for MNIST dataset.

6 Conclusion

In this work, we have proposed a viable alternative to traditional averaging techniques for aggregating logits from multiple columns. The proposed approach analyzes weakly learnt concepts and guides the backpropagation gradients along those pathways to address the weaknesses. Two inference protocols have been proposed that can either focus on the strongest pathway or even average of the improved logits. The proposed model triumphs of the standard averaging technique across 13 different datasets. It evens achieves new benchmark scores in a few datasets. It is also established statistically that for several applications, the learned concepts of the proposed method are significantly different than the competing methods. Currently, the proposed method works only in the output layer with logits, however it may be possible to adopt a similar approach in future works for more inner layers.

References

1. Cireşan, D., Meier, U., Schmidhuber, J.: Multi-column deep neural networks for image classification. In: 2012 IEEE conference on computer vision and pattern recognition. pp. 3642–3649. IEEE (2012)
2. Cireşan, D., Meier, U., Masci, J., Schmidhuber, J.: Multi-column deep neural network for traffic sign classification. Neural networks 32, 333–338 (2012)
3. Ghosh, S., Das, N., Das, I., Maulik, U.: Understanding deep learning techniques for image segmentation. ACM Computing Surveys (CSUR) 52(4), 1–35 (2019)
4. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., Liu, T., Wang, X., Wang, G., Cai, J., et al.: Recent advances in convolutional neural networks. Pattern Recognition 77, 354–377 (2018)
5. He, K., Zhang, X., Ren, S., Sun, J.: Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 770–778 (2016)
6. Krizhevsky, A., Sutskever, I., Hinton, G.E.: Imagenet classification with deep convolutional neural networks. In: Advances in neural information processing systems. pp. 1097–1105 (2012)
7. LeCun, Y., Bottou, L., Bengio, Y., Haffner, P.: Gradient-based learning applied to document recognition. Proceedings of the IEEE 86(11), 2278–2324 (1998)
8. Litjens, G., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., Van Der Laak, J.A., Van Ginneken, B., Sánchez, C.I.: A survey on deep learning in medical image analysis. Medical image analysis 42, 60–88 (2017)

9. Mandal, B., Sarkhel, R., Ghosh, S., Das, N., Nasipuri, M.: Two-phase dynamic routing for micro and macro-level equivariance in multi-column capsule networks. Pattern Recognition 109, 107595
10. Redmon, J., Divvala, S., Girshick, R., Farhadi, A.: You only look once: Unified, real-time object detection. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 779–788 (2016)
11. Redmon, J., Farhadi, A.: Yolov3: An incremental improvement. arXiv preprint arXiv:1804.02767 (2018)
12. Sarkhel, R., Das, N., Das, A., Kundu, M., Nasipuri, M.: A multi-scale deep quad tree based feature extraction method for the recognition of isolated handwritten characters of popular indic scripts. Pattern Recognition 71, 78–93 (2017)
13. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556 (2014)
14. Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., Rabinovich, A.: Going deeper with convolutions. In: Proceedings of the IEEE conference on computer vision and pattern recognition. pp. 1–9 (2015)
15. Ukil, S., Ghosh, S., Obaidullah, S.M., Santosh, K., Roy, K., Das, N.: Improved word-level handwritten indic script identification by integrating small convolutional neural networks. Neural Computing and Applications pp. 1–16 (2019)
16. Ukil, S., Ghosh, S., Obaidullah, S.M., Santosh, K., Roy, K., Das, N.: Deep learning for word-level handwritten indic script identification. Proceedings of the 3rd International Conference on Recent Trends in Image Processing and Pattern Recognition(RTIP2R 2020) (2020)