Incremental Learning of Object Detector with Limited Training Data

Muhammad Abdullah Hafeez, Adnan Ul-Hasan and Faisal Shafait
School of Electrical Engineering and Computer Science (SEECS)
National University of Sciences and Technology (NUST)
Islamabad, Pakistan
Deep Learning Laboratory, National Center of Artificial Intelligence (NCAI)
Islamabad, Pakistan
{mhafeez.mscs16seecs, adnan.ulhassan, faisal.shafait}@seecs.edu.pk

***Abstract*—State of the art Deep learning models, despite being at par to the human level in some of the challenging tasks, still suffer badly when they are put in the condition where they have to learn with time. This open challenge problem of making deep learning model learn with time is referred in the literature as Lifelong Learning, Incremental Learning or Continual Learning. In each increment, new classes/tasks are introduced to the existing model and trained on them while maintaining the accuracy of the previously learned classes/tasks. But accuracy of the deep learning model on the previously learned classes/tasks decreases with each increment. The main reason behind this accuracy drop is catastrophic forgetting, an inherent flaw in the deep learning models, where weights learned during the past increments, get disturbed while learning the new classes/tasks from new increment. Several approaches have been proposed to mitigate or avoid this catastrophic forgetting, such as the use of knowledge distillation, rehearsal over previous classes, or dedicated paths for different increments, etc. In this work, we have proposed a novel approach based on transfer learning methodology, which uses a combination of pre-trained shared and fixed network as a backbone, along with a dedicated network extension in incremental setting for the learning of new tasks incrementally. The results have shown that our approach has better performance in two ways. First, our model has significantly better overall incremental accuracy than that of the best in class model in different incremental configurations. Second, our approach achieves better results while maintaining properties of true incremental learning algorithm i.e. successful avoidance of the catastrophic forgetting issue and complete eradication of the need of saved exemplars or retraining phases, which are required by the current state of the art model to maintain performance.**

***Index Terms*—Incremental Learning, Continual Learning, Lifelong learning, Deep Learning, Transfer Learning.**

I. Introduction

Currently, deep learning models have outperformed other ML model families in various domains. In recent years, Deep Learning (DL) models have advanced than any other family of machine learning models. After the AlexNet [3] success on dataset from ILSVRC-2010, and iLSVRC-2012 [4] many state of the art deep learning architectures like Resnet [5], GoogleNet [6], Inception [7], and Densenet [8], have been proposed. These state of the art architectures has performed extremely well on a vast variety of datasets in different domains like Image recognition, Object detection, Object recognition, Natural Language Processing, etc. Deep learning models have even surpassed humans in performance in specific settings.

Despite being outperforming and powerful models, their design has some limitations. Most prominent is that data used for training a DL model must be available right at the time of training. But in real-world, it is not always possible to have all the training data available during the training phase. This raises the requirement of DL models to a must-have feature of able to expand their knowledge with the passage of time incrementally i.e. when the new learning data becomes available. Class/Task Incremental learning model, thus is a model capable of extending its knowledge with the passage of time by learning new classes/tasks, i.e., when the training data of new classes/tasks become available to the model, without effecting its performance on previously learned classes/tasks. See the Figure 1. Characteristics of Incremental learning model will be discussed later in this section. Challenges involved in designing DL models capable of incrementally learning with time had been studied since the early 90s. Although DL models are very flexible in design from one perspective (plastic nature of ANNs) but there are some inherent issues with their design that makes it difficult to work around and make them fully incremental learning models.

One of the main issues for DL models to have true incremental learning capability is Catastrophic Forgetting. By design DL models have learn-able parameters called weights. These weights are learned (adjusted) iteratively during the training phase. These learned weights collectively depict the approximation of function that the model has learned during the training phase. Therefore in the incremental setting when the new data becomes available to the model, it tries to learn on newly available data. Where in this training process, the weights of the models adjust themselves (plastic nature of weights) according to this new data and hence the adjustment of weights for the previously learned data (stability of weights) gets disturbed. As a result model performance on the previously learned tasks/classes starts to degrade while learning the new ones. This issue of adjustment of weights again and again for each new increment and resulting in degradation of performance over previous classes/tasks is referred as

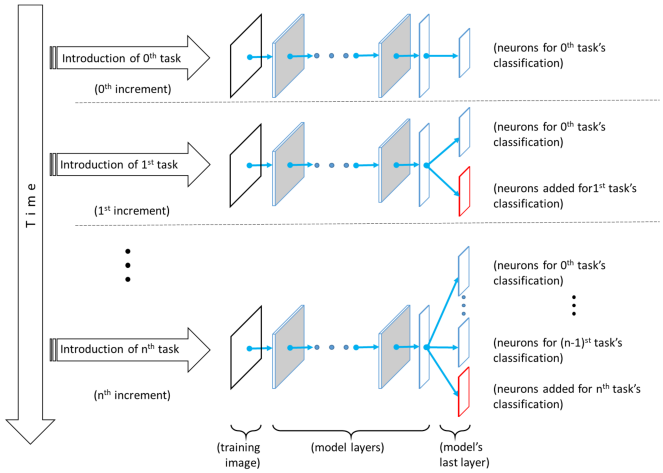

Fig. 1: General schematic of Incremental learning algorithm: Where model learns new classes as introduced with the passage of time, and can classify all the classes trained so far.

Catastrophic Forgetting/Interference [30], [31]. Whereas in DL models, this balance between plasticity (ability to learn new tasks/classes) and stability (ability to retain the performance on the old tasks/classes) is called the stability-plasticity dilemma.

Formally, for a DL-model to be classified as true Incremental learning model, we seek following characteristics to be met by the model. I. Model should be resistant to Catastrophic Forgetting, so that it's performance on the previously learned classes should not degrade at all or at least retain substantially. II. Ideally model should be independent of rehearsal phases, i.e., model should not require any retraining over that past learned classes/tasks data for a persistence performance in the incremental setting. III. Model design should be capable to keep on learning and be able to accommodate new classes/tasks with the passage of time i.e. there should be no upper bound for the increment number up to where increments can be added. IV. As the classes/tasks are added to the model, its size increases due to addition of new connections. But model-size growth rate must be very low per increment.

Organization this paper is as follows: Section II discusses the related work done. Section III explains the methodology and its working pipelines in detail. In Section IV, results from our proposed model are reported on CIFAR-100 dataset in four different incremental configurations and showed the improvements over the best in class strategies. In the last in Section V, we conclude our work, with the limitations of work and future work directions.

II. Related work

Incremental Learning is a long-standing challenge in the field of Machine Learning [32] [33], [34]. Many approaches have been devised to achieve true incremental learning capability, both in domain of Deep Learning models as well as in other Machine Learning models. Here we discuss, some prominent strategies under the similarities in their design philosophies.

A. Retraining phase

One design philosophy used in incremental learning models is the use of "Retraining Phase". Where, during the training phase of a new task, some part of the previous tasks' training data is also used with the training data of the new task. In this regard, some mechanism is devised to select some part of the training data form previously trained classes to approximate the their distribution. In iCaRL [9], they used herding technique from [10], to get the best exemplars for the retraining phase. Herding was used in order to achieve two objectives: first, to make exemplars set to exhibit true class means approximation at the initial stage, second, herding algorithm could remove exemplars when necessary while keeping the approximation as close as possible, thus usage of disk space for saving the exemplars was kept constant.

Another approach for the retraining phase was the use of Generative adversarial networks [11], [12]. In this setting training data of old classes were generated with the use of generator of the generative adversarial network and thus augmented with the training data for the new increment. In [13] they used the combination of generator and solver, making a complete model for the current state. On new increment, the generator and solver of old scholar generated pseudo training samples for old data and their labels respectively, which were then combined with the data and labels for new classes to train the new scholar. The advantage of this generative adversarial network-based retraining approach is retaining the accuracy over the old classes along with learning the new classes while eliminating the requirement of saving old classes' data on disk.

Limitation of this method is that generator has difficulty in generating visually complex dataset classes like CIFAR100 [14], CUBs200 [15], ImageNet [4] etc, as mentioned in [16]. Therefore performance drops when generator based architecture is put to test with such datasets. This results in limiting its application over the visually complex datasets.

B. Expansion of Network

In incremental setting, every time in each new increment, model is further trained on some specific number of new classes. Which requires the addition of output nodes in the output layer of the network. Therefore, in iCaRL [17] when new nodes are added to the network, a new set of weights making the connection between output and last hidden layer are also added

In Learning without forgetting [18], they used the task incremental strategy instead of class incremental strategy. Where their network was divided into two categories depending on the parameters' access to the specific task. One is a shared part of the network that has shared parameters θ_s across all tasks, another part is network extension with parameters θ_n which was added on the introduction of a new task. While at the same level of θ_n, parameters of network extensions added for old tasks during previous increments, were referred as θ_o.

Therefore, it is inevitable that an incremental model will be extended every time a new increment is added to the model.

C. Distillation loss

During the training phase of the new increment, the retraining part is required. Where some mechanism of loss calculation over the previous training dataset becomes necessary, in order to maintain some balance between the training quality of new classes and retaining the accuracy over old classes. This is usually done using the changes in the loss calculation method in the loss function for the model. In [17] they used the loss function as the combination of two-loss functions: classification loss and distillation loss. The latter term was used to prevent Catastrophic Forgetting by using the variation of loss function used in [16]. The equation used by [17] for the calculation of loss during training was originally proposed for the information transfer between neural networks in [35]. But in [17] it was used in different time stamps in the same neural network.

The same technique of Distillation loss proposed in [35], was used by [18] for the retention of accuracy over the old tasks after the addition of a new task. But in [18] they used it for task incremental scenario rather class incremental scenario as used in [17].

D. Nearest Class Mean Classifier

In [19], it was exhibited that the nearest-class-mean classifier can incrementally accommodate new classes, by keeping the mean of feature vectors of all examples form all classes. During the testing phase, the feature vector of test example was classified with the label of mean-of-feature-vectors, which was most similar on metric to feature vector of test example. This proposed method exhibited to work well when applied in the incremental learning scenario [19], [20], [17].

iCaRL [17] despite being Deep learning based architecture, they did not used neural network based classification layer for final classification. Instead, they used different family of ML-algorithm for the classification purpose, along with the neural network as the backbone of the whole algorithm. They used the nearest-mean-of-exemplars as their classification algorithm.

The main idea of using this algorithm for classification was taken from the nearest class-mean classifier from [19]. The difference between both implementations [17] and [19] is the use of only feature representation of exemplars in [17], instead of feature representation of whole data set examples as in [19]. This is due to the reason that they cannot have all the training data stored for the above-mentioned purpose in the incremental setting which will consume the disk space with time. Usage of this algorithm in [17] made the classifier robust to changes in feature representation, as class-prototypes could automatically adjust themselves to changes in feature representation.

E. Subnetworks in Network

Inspired by pruning techniques in neural networks, and redundancies in large deep learning models, PackNet [21] proposed the method for learning of tasks with time in an incremental manner, which is relatively different than all the other models discussed here yet. They used the redundancies of the deep learning model to their advantage and employed weight-magnitude-based pruning methodology introduced in (cite- A.Mallya 7,8) and freed up the excess parameters for the learning of new tasks without the drop of performance in any task learned. Using network retraining along with iterative pruning they were able to make the model learn multiple tasks with minimal drop in performance. Another major difference from other tasks was that they always used to optimize for the current (new) task, rather seeking a performance balance between the new and old tasks' learning using some proxy loss functions.

The limitation of PackNet [21] is that it cannot keep on adding new tasks with time, a property, a must have property of a true incremental learning algorithm as mentioned in point III in Section 1. This is due to the reason that after some increments there will be no more room left for pruning in the model thus model will reach to its limit and will not be able to learn any further tasks afterward.

III. Methodology

The basic design of our proposed model is based on the design philosophy of avoiding Catastrophic Forgetting in the first place. In this way, a chain of issues involved with m̈inimizing the effect of catastrophic forgetting" approach can be avoided as well. For example in PackNet [21], they used the similar design approach where catastrophic forgetting was not an issue. Therefore, during the increment, they only focused on optimizing for the task in hand, rather to worry about retraining phases and all complexities involved along with that phase.

In our work, complete proposed model can be divided into the following three components:

- Base model: It is based on multiple layers of Convolutional Neural Network (CNN) architecture working as the backbone of the model.
- Network Extension (NE): It is the extension layer based on CNN architecture, which is augmented at the end of the network for incremental learning.
- Final classification: In this component of the model fully connected layers from all network extensions are combined to get the final classification.

A. Base model

The architecture backbone for our proposed model is one of the state of the art deep learning model named ResNet34 [5] with some modifications. ResNet34 architecture's first layer is a convolution layer followed by 32 bottleneck-blocks, while the FC layer is at the end of last bottleneck-block for final classification. These bottleneck-blocks are grouped together and named as layer. Thus total of four groups of 6, 8, 12, and 6 bottleneck-blocks form these intermediate layers. ResNet34 has 3.6 billion FLOPS. In the proposed model here, we have used the ResNet34 network per-trained on the ImageNet [4] 1000-class dataset. This pretrained part of the network

will be shared across all increments and will remain fixed. Therefore none of the training phase of any increment will change the parameters of this shared part of the model. In the proposed model, fully connected (FC) layer of the pretrained ResNet34 is removed and network is extended with a block of convolution layer and a fully connected layer of required size on each increment.

B. Network Extension

In the proposed model, fully connected (FC) layer of the pretrained ResNet34 is removed and network is extended with a block of convolution and a fully connected layer of required size on each increment. We call this additional block as Network Extension (NE). The structure of the NE has convolution layer of 960 input and 512 ouput channels, siz of the kernel used is 3x3. Convolution layer is followed by batch normalization layer and ReLU activation function respectively. In the last fully connected (FC) layer is attached, which has 512 input neurons with a 0.5 dropout probability and output neurons are equal to the number of classes in the current increment. Softmax activation function is used on the final output. When the training data is passed to the model, output features from all four intermediate layers of the ResNet part are taken in order to get all levels of features i.e. from high level to low-level. Spatial size (height x width) of the outputs from first, second, and third intermediate layers are 56x56, 28x28, and 14x14 respectively. These outputs are then resized to 7x7 using downsampling method and then concatenated with the output of the fourth intermediate layer which has already output of size 7x7. This combined output of size 7x7x960 (height x width x number-of-feature-maps) from all intermediate layers are then passed to the Network Extension.

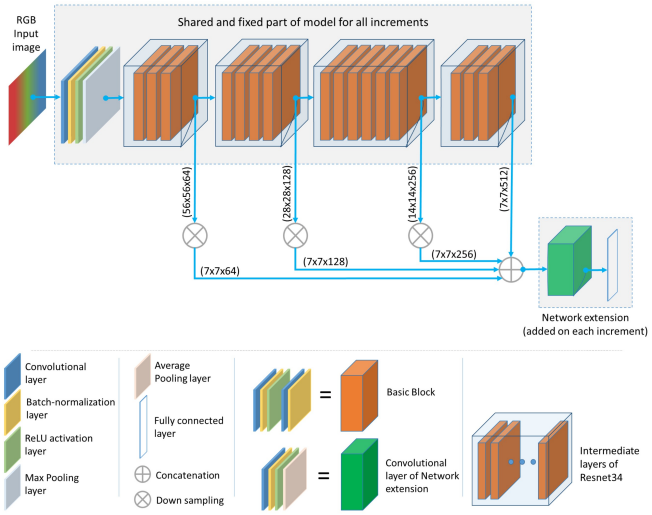

Fig. 2: Architecture of Proposed model: Shared and fixed layers of the model are taken from the pre-trained Resnet34. While Network Extension part of the model has one dedicated Convolutional and Fully connected layer, which are added for each increment, during the training phase of the incremental setting.

C. Final Classification

A fully connected layer in each network extension gives output for all passed samples i.e. both in-distribution and out-of-distribution samples. It was observed that, most of the time, for same test sample, output logit value generated from its relevant NE is relatively high than logit values generated from all other irrelevant NEs. Therefore, the idea is to use these logits produced for all samples from all the network extensions and then stack them sample-wise and use the max function i.e. torch.max() along the stacked dimension. Thus output of the max function is used for final classification.

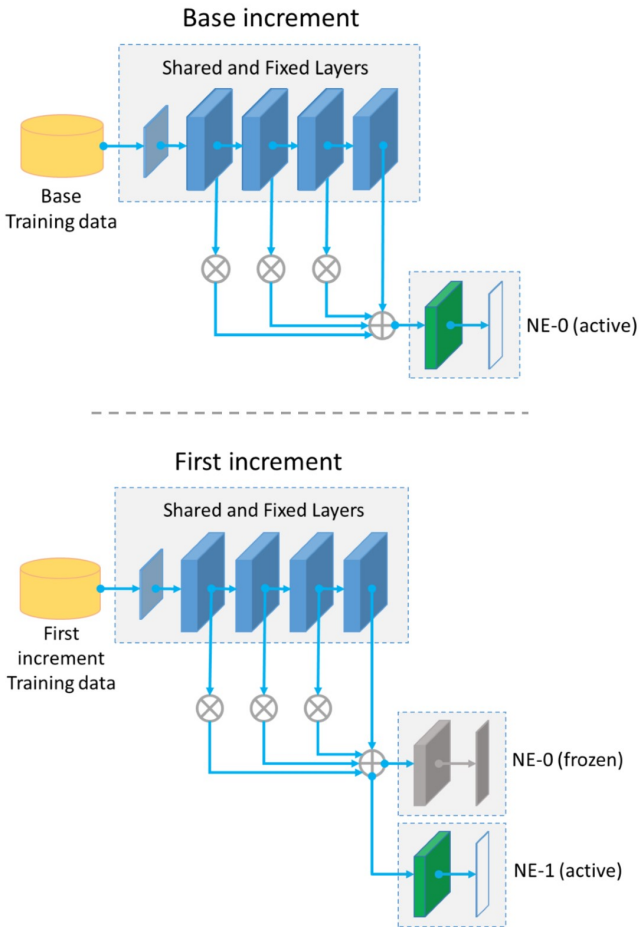

Fig. 3: Training pipeline of model: Two increments are shown i.e. Base increment, first increment. Legend is in Fig 3.5

$$y'_f = argmax\Big(concat\Big[y'_i\Big]_{i=0}^{n}\Big) \tag{1}$$

where:

y'_i = logits of i^{th} Network Extension (NE)
n = Total number of increments at specific time stamp
$concat()$ = concatenation function: concatenates logits in specified dimension
$argmax()$ = function to get the index of max-value in specified dimension

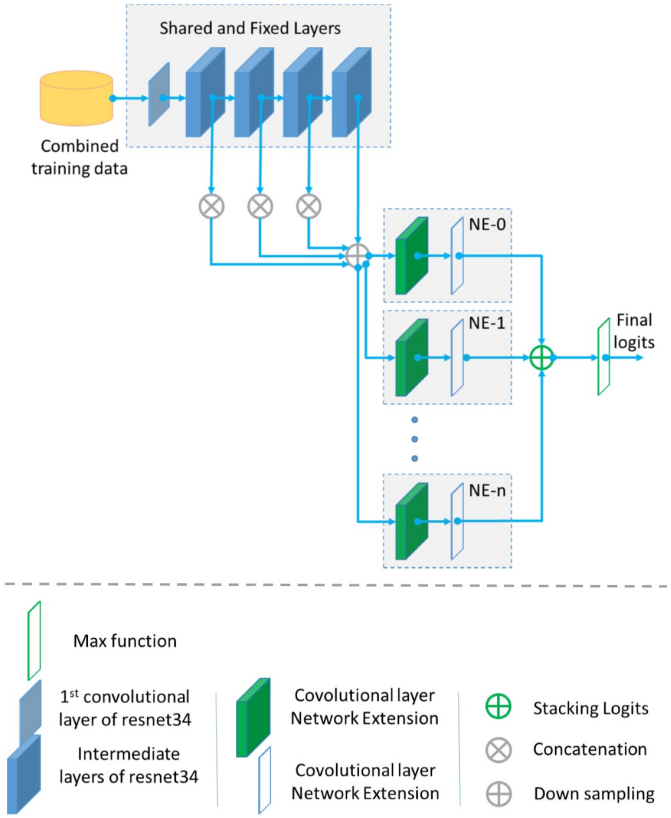

Fig. 4: Testing pipeline of model: Combined testing for 'n' increments. Data passed is combined testing data set from all 'n' increments

D. Training phase

During the training phase, the shared part of the model remains fixed as discussed previously. However, for each increment right from zeroth increment (base increment), network extension will be added for the increment under training. While network extensions from all previous increments will not be accessible to the training phase of current increment i.e. their parameter will remain unchanged. Also NEs from all previous increments will not participate in the training phase of the current increment i.e. they will not participate in the calculation of final logit value of current increment. With this setting, the whole model will then be trained on the dataset of the current increment only and will be optimized for the current training dataset. No retraining or dataset from old increments will be required at all in any future increment. Parameters learned for previous increments will be saved on the disk during training phase of current increment.

E. Testing phase

During the testing phase all NEs from all increments will be active. Shared part of the model along with network extensions from all the increments will be loaded to the model in torch.eval() mode. Combined test data from all increments will be passed at once to model in all the network extensions. Each extension will make prediction for both in-distribution and out-of-distribution examples. Prediction from all network extensions will then be combined in the manner specified priviouosly and final classification will be made using max function over the stacked logits from all NEs.

IV. Experiments

Incremental performance of the model, unlike single figure performance measure, is usually evaluated in the incremental manner, by oveserving the overall performance of the model over the course of all increments. Therfore we have compared the performance graphs in four different incremental configurations. This approach depite being a preferable evaluation measure, we have also reported performance of our model in terms of single performance value as Average Incremental Accuracy as used in [7]. Here in this section we compare and evalute our model with best in class and other strategies under inter-model subsection. We also compare intra-model configurations to get the insights on which component helped in the performance improvement under intra-model subsection.

A. Inter-model comparisons

Accuracy comparisons were done in two settings: First use of combined testing data from all increments i.e. all previous and current increments at the time of testing, secondly the use of separate testing data from all increments is used - at each increment in the testing phase. Therefore for the combined testing data, we compared results with state of the art iCaRL [9], and baseline Lwf[11]. While the performance of the proposed model on separate testing data for each increment, results are compared with the results from PackNet[20].

Therefore for the testing incremental accuracy of our model, we used the following four different incremental configurations.

- Configuration-1, we used two increments, 50 unique classes were introduced to the model in each increment.
- Configuration-2, we used five increments, , 20 unique classes were introduced to the model in each increment.
- Configuration-3, we used ten increments, , 10 unique classes were introduced to the model in each increment.
- Configuration-4, we used 20 increments, five unique classes were introduced to the model in each increment.

Results of the proposed model are compared with the state of the art and other implementations are given in Figure 5

TABLE I: Average accuracy improvement over different incremental configurations

Model-name → batch size ↓	LwF	iCaRL	ours (Densenet201 + Single CNN)	ours (Resnet34 + SMNE)
5 classes	32.46	61.33	**62.02**	61.91
10 classes	44.33	64.00	67.87	**68.62**
20 classes	54.40	67.40	70.38	**72.85**
50 classes	64.25	68.50	69.625	**74.91**

From the confusion matrices Generally, true positive rates for all classes are significantly higher than their respective false negative and false positives. Learning of the model for all classes from all increments is optimized.

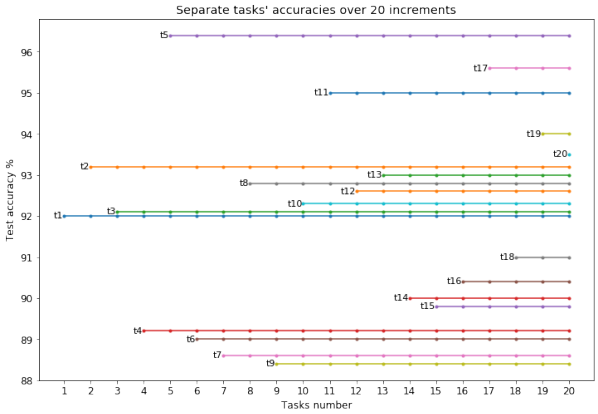

Fig. 5: Accuracy of separate task in separate testing setting

The spread of predictions made by the model for each class is uniformly distributed. The model is unbiased in retaining the performance for all classes and increments

True positive rates in The Matrix where the number of increments is less has generally higher true positive rates than matrix where the number of increments is higher. Less increments have better overall performance than more increments, although the total number of classes learned are the same.

Optimized Accuracy for each task. Due to the independent NE for each task.

Accuracy of each task is sustained for all the succeeding increments, at the point where it was during the training time. Catastrophic forgetting is successfully avoided in the model. Shared parameters are fixed. Therefore weights do not get disturbed during the training phase of the new increment.

By design, the proposed model is capable of sustaining accuracies for extended increments. Scale-able and demonstrated sustained accuracies up to 20 increments whereas PackNet is not scale-able and demonstrated for only 5 increments

B. Intra-model comparisons

1) Base models: The pre-trained model is used as a base in the proposed model for general feature extraction and was supposed to help in improving performance along with avoiding catastrophic forgetting.

Experimentation showed that usage of a better pre-trained model as a base, actually helped in improving the overall performance.

As it gives better out of the box feature extractor, upon which further training is based

TABLE II: top errors of different pre-trained models used as base in proposed work

Base model	Top-1 error	Top-5 error
Resnet 18	30.24	10.92
Resnet 34	26.70	8.58
Densenet 201	22.80	6.43

2) Network extensions: In order to maximize the performance output while keeping the network size at a minimum, we used three different configurations as Network extension. For all the network extension configurations base model was fixed to Resnet-34 architecture and the incremental configuration was batch of 5 classes for 20 increments. details of network extensions are as follows:

- Single CNN layer: This layer was used at the end of the last CNN layer of the base model. Where In and out channels were 960 and 512 respectively. Batch normalization was used. whereas ReLU is used as an activation function
- Inception block: This layer was used at the end of the last CNN layer of the base model. Inception v1 configuration 960, 96, 96, 256, 32, 96, 64 as In, 1x1, 1x1, 3x3, 1x1, 5x5, pool was used.
- Sequenced MobileNet Extensions (SMNEs): MobileNet v1 configuration: Depth wise Conv and pointwise conv block. Each block is added after each intermediate layer Concatenated with the succeeding layer output Then passed to the next block

TABLE III: Network Extensions comparisons

Network extension	parameters	NE to model ratio	NE size reduction	Accuracy improved
Single CNN	4,427,269	20.31	-	-
Inception Block	529,157	2.42	88.02 %	-1.7 %
SMNEs	**446,565**	**2.05**	**89.91 %**	**2.34 %**

C. Discussion

In the proposed work, as discussed in the Methodology section, for the final classification, we have used the *argmax* function over the stacked logits from all Network Extensions. Factors involved in the success of this very simple and straight forward method are as follows:

I First, argmax() function alone cannot be credited for the performance improvement. The whole design of the model itself, training, and the testing pipeline is actually aiding argmax() function in achieving the presented performance.

II In model design, shared parameters remain fixed for all the increments. Therefore each increment utilizes the shared parameters during training but cannot update them during backpropagation.

III During the training phase, network extensions form all previous increments are neither accessible nor involved in the training process of the current increment.

IV Training data form previously trained classes are not passed to the model except the current task's own training data.

V Training phase of the current task is totally independent of all other tasks' training. Therefore model training is focused on the current task's training to achieve the best accuracy, rather keep accuracy balance between current and old tasks.

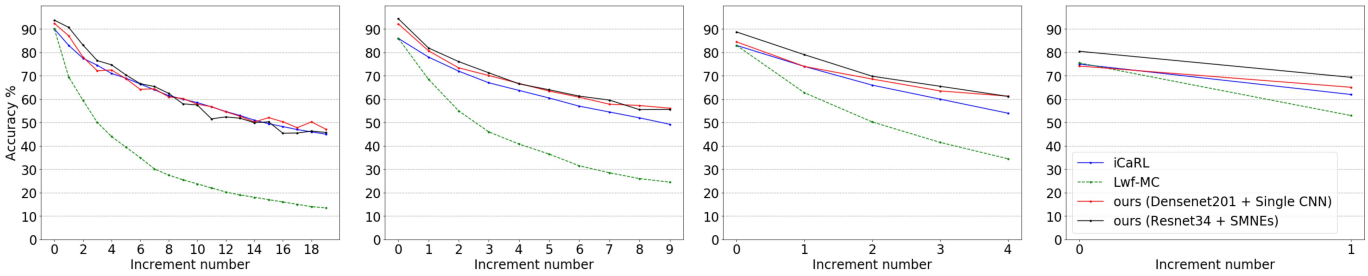

Fig. 6: Incremental Accuracies in configuration 1, 2, 3 and 4 from right to left

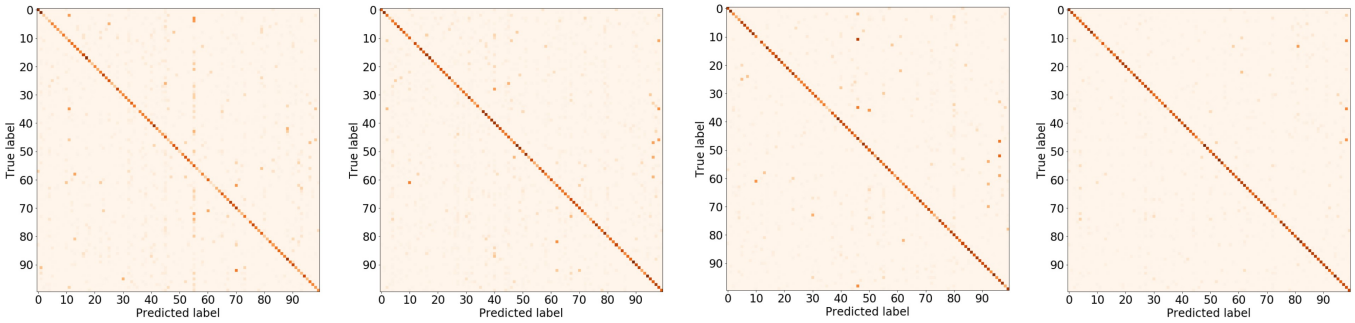

Fig. 7: Confusion Matrix in configuration 1, 2, 3 and 4 from right to left

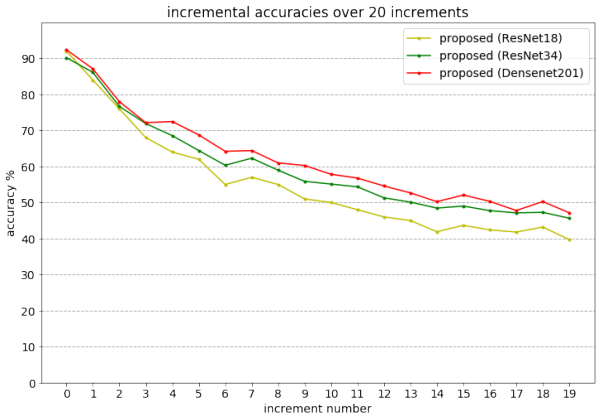

Fig. 8: Accuracy comparison with different base models

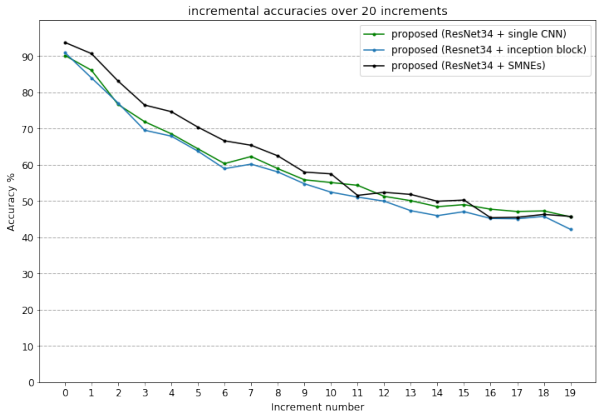

Fig. 9: Accuracy comparison of different Network extensions

VI In each increment, the softmax() function is used. Which keeps the loss to a minimum by keeping the relevant logit value as high as possible while irrelevant logit values to as low as possible.

Therefore, keeping in view of the above-mentioned points, the training phase of new increment is completely independent: in terms of parameters (point (II) and (III), in terms of training data (point (IV)), and in terms of optimization (point (V)). Being completely independent in the training phase and usage of softmax() outputs in loss calculation result in a high difference between relevant and non-relevant logits.

Therefore, during the testing phase when combined testing data from all tasks are passed to the model. Logit value from relevant Network Extension for a relevant class is most of the time higher than all other logit values produced from other non-relevant Network extensions. Due to which argmax() function over the stacked logit values performs significantly better.

V. Conclusion

We have introduced the model for Incremental learning which learns to classify the different classes as introduced to the model in an incremental fashion with the passage of time. Model architecture can be divided into three main parts: (1) Pre-trained part which is shared and remains fixed across

all increments. (2) Network extension, trainable part of the model only for relevant increment and is not accessible to other increments in the training phase. It is added for each new increment (3) During the testing phase, the argmax() function is applied to the stacked logits from all Network Extensions.

Experiments performed on CIFAR-100 showed that the proposed model can learn incrementally for an extended period of time and outperformed best in the class model by a significant margin without any requirement of retraining phase or saving of old data, a characteristic property of an ideal incremental learning model. This is achieved by designing the model on the philosophy of avoiding rather minimizing the Catastrophic forgetting.

Although the Proposed model has outperformed best in class results while maintaining the properties of a true incremental learning model. Still, the overall performance has a significant improvement gap. In the future, we have plans to introduce the Out-of-distribution techniques and employ the strategies to minimize the Network extension size as well.

REFERENCES

[1] J. Fagot and R. Cook. Evidence for large long-term memory capacities in baboons and pigeons and its implications for learning and the evolution of cognition. Proceedings of the National Academy of Sciences, 103(46):17564-17567, 2006.

[2] W. Abraham and A. Robins. Memory retention the synaptic stability versus plasticity dilemma. Trends in neurosciences, 28(2):73-78, 2005.

[3] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems, pages 1097-1105, 2012.

[4] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. FeiFei. ImageNet Large Scale Visual Recognition Challenge. International Journal of Computer Vision (IJCV), 115(3):211-252, 2015.

[5] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770-778, 2016.

[6] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1-9, 2015.

[7] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. arXiv preprint arXiv:1502.03167, 2015.

[8] G. Huang, Z. Liu, L. van der Maaten, and K. Weinberger. Densely connected convolutional networks. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[9] SA. Rebuffi, A. Kolesnikov, G. Sperl, and C. Lampert. icarl: Incremental classifier and representation learning. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), July 2017.

[10] M. Welling. Herding dynamical weights to learn. In Proceedings of the 26th Annual International Conference on Machine Learning, pages 1121-1128, 2009.

[11] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, Advances in Neural Information Processing Systems 27, pages 2672-2680. Curran Associates, Inc., 2014.

[12] A. Radford, L. Metz, and S. Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks, 2015.

[13] H. Shin, J. Lee, J. Kim, and J. Kim. Continual learning with deep generative replay. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, Advances in Neural Information Processing Systems 30, pages 2990-2999. Curran Associates, Inc., 2017.

[14] A. Krizhevsky, G. Hinton, et al. Learning multiple layers of features from tiny images. 2009.

[15] P. Welinder, S. Branson, T. Mita, C. Wah, F. Schroff, S. Belongie, and P. Perona. Caltech-UCSD Birds 200. Technical Report CNS-TR-2010-001, California Institute of Technology, 2010.

[16] J. Kim, J. Kim, and N. Kwak. Stacknet: Stacking feature maps for continual learning. In The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops, June 2020.

[17] M. Ristin, M. Guillaumin, J. Gall, and L. Van Gool. Incremental learning of ncm forests for large-scale image classification. In The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), June 2014.

[18] Z. Li and D. Hoiem. Learning without forgetting. IEEE transactions on pattern analysis and machine intelligence, 40(12):2935-2947, 2017.

[19] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. In A. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, editors, Computer Vision - ECCV 2012, pages 488-501, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[20] T. Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Distancebased image classification: Generalizing to new classes at near-zero cost. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35(11):2624-2637, 2013.

[21] A. Mallya and S. Lazebnik. Packnet: Adding multiple tasks to a single network by iterative pruning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 7765-7773, 2018.

[22] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. IEEE Transactions on Neural Networks, 5(2):157-166, 1994.

[23] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Proceedings of the thirteenth international conference on artificial intelligence and statistics, pages 249256, 2010.

[24] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. In B. Leibe, J. Matas, N. Sebe, and M. Welling, editors, Computer Vision - ECCV 2016, pages 630-645, Cham, 2016. Springer International Publishing.

[25] G. Cybenko. "Approximation by superpositions of a sigmoidal function." Mathematics of control, signals and systems 2.4 (1989): 303-314.

[26] BC. Csáji, Balázs Csanád. "Approximation with artificial neural networks." Faculty of Sciences, Etvs Lornd University, Hungary 24.48 (2001): 7.

[27] Alom, Md Zahangir, Tarek M. Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C. Van Essen, Abdul AS Awwal, and Vijayan K. Asari. "A state-of-the-art survey on deep learning theory and architectures." Electronics 8, no. 3 (2019): 292.

[28] Huang, Gao, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q. Weinberger. "Deep networks with stochastic depth." In European conference on computer vision, pp. 646-661. Springer, Cham, 2016.

[29] Shams, S., Platania, R., Lee, K. and Park, S.J., 2017, June. Evaluation of deep learning frameworks over different HPC architectures. In 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS) (pp. 1389-1396). IEEE.

[30] R. French. Catastrophic forgetting in connectionist networks. Trends in cognitive sciences, 3(4):128-135, 1999.

[31] M. McCloskey and N. Cohen. Catastrophic interference in connectionist networks: The sequential learning problem. In Psychology of learning and motivation, volume 24, pages 109-165. Elsevier, 1989.

[32] Thrun, S., Mitchell, T. (1995). Lifelong robot learning. Robotics and Autonomous Systems, 15, 25-46.

[33] Robins, A. V. (1993). CaTastrophic forgetting in neural networks: The role of rehearsal mechanisms. In Proceedings of the first new zealand international twostream conference on artificial neural networks and expert systems. IEEE Computer Society Press.

[34] Robins, A. V. (1995). CaTastrophic forgetting, rehearsal and pseudorehearsal. Connection Science, 7(2), 123–146

[35] G. Hinton, O. Vinyals, and J. Dean. Distilling the knowledge in a neural network. In NIPS Workshop, 2014.