LEARNING CONVOLUTIONAL NEURAL NETWORKS WITH DEEP PART EMBEDDINGS

Nitin Gupta[⋆] *Shashank Mujumdar*[⋆] *Prerna Agarwal*[⋆] *Abhinav Jain* *Sameep Mehta*

IBM Research Laboratory, India

ABSTRACT

We propose a novel concept of Deep Part Embeddings (DPEs), which can be used to learn new Convolutional Neural Networks (CNNs) for different classes. We define DPE as a neuron of a trained CNN along with its network of filter activations that is interpretable as a part of a class that the neuron contributes to. Given a new class $\mathcal{C}$, we explore the idea of combining different DPEs that intuitively constitute $\mathcal{C}$, from trained CNNs (not on $\mathcal{C}$), into a network that learns the class $\mathcal{C}$ with few training samples. An important application of our proposed framework is the ability to modify a CNN trained on n classes to learn a new class with limited training data without significantly affecting its performance on the n classes. We visually illustrate the different network architectures and extensively evaluate their performance against the baselines.

Index Terms— Model Learning, DPE, Activation, CNN

1. INTRODUCTION

Convolutional Neural Networks rely on training millions of parameters from large amount of labeled data and computational resources which pose a problem for training a new CNN from scratch. To address this, different methods have been proposed in literature that make use of transfer learning [1–6], knowledge distillation [7, 8], filter pruning [9, 10], life long learning [11–13] etc. to accelerate re-usability of trained CNNs. However, the final network architecture achieved from these methods still operate as a "black-box" with little understanding of the features it learns. Attempts have been made to understand how CNNs work in literature. In [14, 15], a multi-layered Deconvolutional Network is utilized to map the features computed at each layer of the network back to the input pixel space. These methods allow for scrutinizing the learned filter weights and gain intuitive insights that validate the learning of the CNN. The code-base in [16] utilizes these approaches and implements a hierarchical visualization of CNN features for a given image that allows to study different neurons and their dependencies between adjacent layers. The neurons progressively compute complex features from simpler ones at higher convolutional layers which correspond to abstract interpretable features regarded as the "parts" of a specific class.

⋆Authors contributed equally to the work

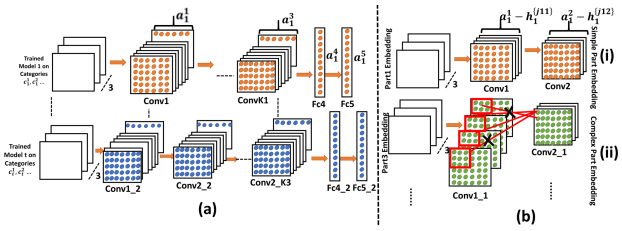

Fig. 1: (a) M_t trained deep models, (b) Examples of simple and complex DPEs extracted from the t trained models; for a complex DPE, the red links show neurons that contribute to the next layer and crosses show the pruned connections.

In this paper, we propose a novel method of training a CNN with Deep Part Embeddings (DPEs). DPE refers to a neuron in higher convolutional layers of a trained CNN along with its sub-network of filter activations starting from the first convolutional layer, that is visually interpretable as a part of the class of interest. For e.g. in Fig. 3(b), we show DPE for part "wheel" of Car class. Given a new class with limited training data and we need to train a new model for it, we showcase different ways of learning interpretable DPEs corresponding to the new class from CNN network(s) not trained on the new class. For e.g. in Fig. 3(c), in order to train a new model for the Bus class with limited samples, we borrow the DPEs that intuitively constitute a bus such as "wheels", "windows" etc. from a CNN network not trained on bus images. We combine these DPEs in a network and train it using limited bus training data to effectively learn the class with high accuracy. We propose two methods of learning - (i) *sequential*; when DPEs are sourced from different CNN architectures and (ii) *shared*; when DPEs are sourced from same CNN architecture.

We extend this approach to the case where we have a trained CNN for n classes and we want to learn a new class with limited training data. We sequentially learn the DPEs for the new class with CNN and retrain with labeled samples for the new class and a subset of the data from n classes. The learnt network is able to achieve high accuracy on new class without significantly affecting the accuracy of n classes.

In summary, the main contributions of this paper are
1. We propose a novel concept of Deep Part Embeddings.
2. We propose a novel methodology to train a CNN for a new class using DPEs, that intuitively constitute the class, with limited training data.

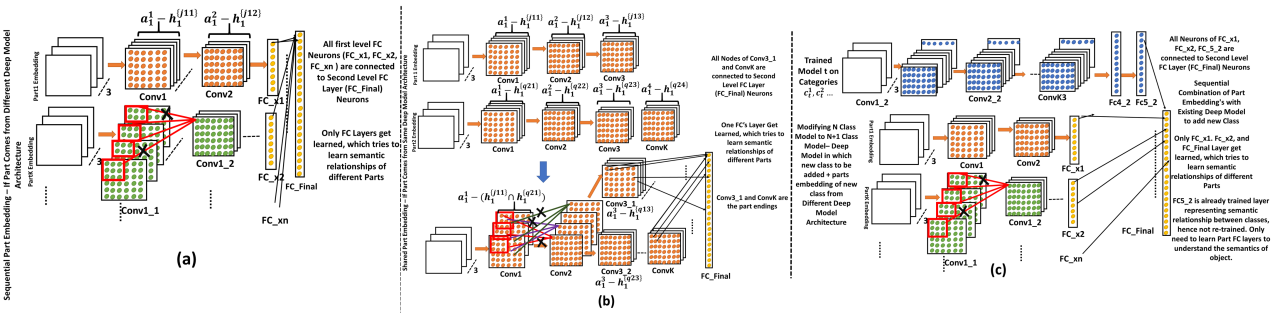

Fig. 2: Illustration of proposed (a) Sequential Learning, (b) Shared Learning Approach, and (c) Architecture that sequentially learn existing model architecture with simple and complex DPEs with Reduced FC layers

3. We propose two methods of learning the DPEs in the CNN architecture when sourced from different CNN architectures (*sequential*) and same CNN architecture (*shared*).
4. We provide a methodology to share filters in case of shared learning architecture and selectively prune filters based on their contribution to the DPEs.

2. PROPOSED METHODOLOGY

2.1. Problem Statement

Let $M = \{M_1, ., M_t\}$ be the set of t trained deep models on different datasets. Let c_i^j represent the j^{th} class for i^{th} deep model, where $i \in \{1, ., t\}$. Our goal is to first extract DPEs from these models and then use the extracted DPEs for training a new CNN with limited number of training samples. We present the details of these methods in the following sections.

2.2. Extraction of Deep Part Embeddings

Fig. 1(a) shows the t deep models with different architectures trained on different classes. For each of these t models, we identify all the neurons that are interpretable as parts of a specified class that the model is trained on. This is achieved by analyzing the hierarchical visualization [16] obtained for different training images of the specified class. For example in Fig. 3 (a), "Wheel" part is represented by three neurons (203, 328, 364) at Conv4 layer. We also use this information to find which of the lower layer neuron(s) contributed in activating the identified neuron(s). For example, if a part x_i^{jk} of some class c_i^j (where k is the total number of parts in class c_i^j) is represented at "Conv3" of a neuron, we store and use this information to identify the neuron(s) which activated it at lower layers i.e., "Conv2" and "Conv1" using the hierarchical representation proposed in [16]. For example, in Fig. 3 (b), the DPE for "Wheel" comprises of three neurons at "Conv4" layer, while rest of the neurons at layer "Conv3", "Conv2", and "Conv1" contribute for the activation of these three neurons at "Conv4". Collectively, these selected neurons along with the underlying network connections among different layers create the DPE of the selected part (in this case "Wheel"). This approach of neuron selection at lower layers relates a feature in layer l to layer $l-1$ using the below weight matrix:

$$w^{l-1,r} = |b^{l-1} \odot \frac{\partial f(b_r^l)}{\partial b^{l-1}}| \quad (1)$$

where, f : feature map $\rightarrow$ R, where R be the sum of all pixels in an image, b_r^l and w_r^l are the activations and weight matrix at layer l for r^{th} feature map. We select those neurons from $l-1$ layer whose values are greater than $(max(|w^{l-1}|) - min(|w^{l-1}|))/2$. At each layer l we identify the set of neurons which do not qualify the constraint as h_i.

Let us assume that $A_i = \{a_i^1, .., a_i^{N_i}\}$ is the architecture of model M_i, where a_i^l represents the set of nodes in l^{th} deep layer and N_i is the total number of layers in M_i. Then, the k^{th} DPE for architecture A_i for class c_i^j can be represented as $P_i^{jk} = \{p_i^{(j)(k)(1)}, ..., p_i^{(j)(k)(l)}..\}$, where $p_i^{(j)(k)(l)} = \{a_i^l - h_i^{(j)(k)(l)}\}$, and $h_i^{(j)(k)(l)}$ represents the set of neurons at l^{th} layer for k^{th} part of j^{th} class in i^{th} model that do not contribute to the selected neurons at $(l+1)^{th}$ layer. Figure 3(a), shows the visual illustration of some of the neurons selected for different parts of Car and Train Class. Figure 3(b) shows the final extracted neurons from each layer, which represents the DPE for the part "Wheel". When we have all the selected neurons at layer l contributing to the selected neurons at layer $(l+1)$, we refer to it as simple part embedding (see Fig. 1(b)(i)).

However, there can be another case in which complex part embedding is possible. In this case, neurons at $p^{(j)(k)(l-1)}$ may not be contributing to all the neurons at $p^{(j)(k)(l)}$ (see Fig. 1(b)(ii)). This can occur due to the pruning constraint applied for neuron connections at each layer. Thus, some neurons at layer $(l-1)$ would only contribute to certain neurons at layer l. In such a case, we break the connections between those nodes while creating a part embedding (see Fig. 1(b)(ii)).

2.3. Sequential Deep Architecture Learning (SeqDPE)

SeqDPE helps to learn the DPEs which are coming from different models. For eg, for the case of two models (M_u and M_v): (a) M_u model contains c_u^j (say "Car") as one of the training

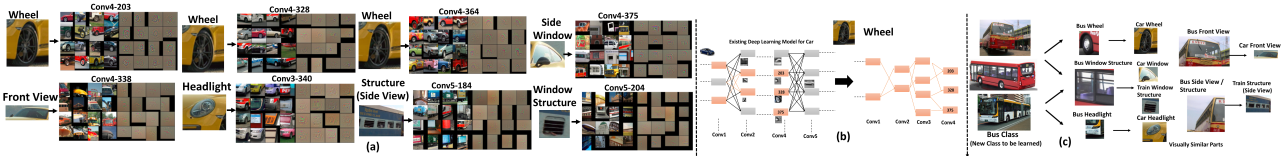

Fig. 3: (a) Representative Neurons of parts for Car and Train Class in Alexnet; (b) DPE for part "Wheel", (c) Parts representation for new class "Bus" to be learned with "Train" and "Car" parts.

Table 1: Comparison of class-wise accuracy (%) for learning a new class with different approaches

	Billiards	Hammock	Horse	Ladder	MotorBikes	T-shirt	Airplanes	Face-Easy	Bus (new Class)
ETE_8CM	62	45.75	49.36	30	97.29	72.83	100	100	-
ETE-25	43.33	12.5	42.10	43.33	96.24	62.5	85.71	96.29	28.53
ETE-50	42.13	18.5	44.10	41.33	97.14	66.25	85.71	96.29	30.53
ETE-150	43.33	17.5	49.10	44.16	96.24	67.81	85.71	96.29	34.53
ETE-200	41.23	41.75	54.89	41.22	96.24	62.5	92.71	100	57
ETE-250	44.27	43.75	57.89	48.33	96.24	65.5	95.71	100	65
FT-25	60	37.5	31.57	26.6	92.59	70.83	95.23	96.29	55.7
Ours (CFC)-25	60.02	43.2	47.24	30	97	70	100	97	74.2
Ours (RFC)-25	61	44.75	48.24	36	97.29	71.24	100	100	77.6

class; (b) M_v model contains $c_v^{j'}$ (say "Train") as one of the training class. For learning a new model M_{t+1} for class c (say "Bus"), we need DPEs of c_u^j and $c_v^{j'}$, as shown in Fig. 2(a).

In SeqDPE we add: (a) first level fully connected (FC) layer - FC layer for each DPE, and (b) second level fully connected layer - FC layer which combine the first level FCs to learn the semantic arrangement of different parts to represent a new class. We illustrate this in Fig. 2(a). The second level FC layer is finally connected to the Soft Max loss layer.

2.4. Shared Deep Architecture Learning (ShaDPE)

For the DPEs extracted from same model architecture, we propose the method of "Shared Learning". Given a model (M_y) that has c_y^j (say "Car") as one of the training class and c_y^k (say "Train") as another training class. For learning a new model M_{t+1} for class c (say "Bus"), we utilize the DPEs of c_y^j and c_y^k, as shown in Fig. 2(b).

In ShaDPE we have two scenarios, (a) all DPEs end at the same layer (say Conv3), (b) DPEs end in different layers (say first DPE ends at Conv3, and second at ConvK as shown in Fig. 2(b)). For scenario (a), we first combine the neurons at each layer while pruning the extra connection between the edges. For scenario (b), we identify the DPE with the least penetration in the network (smallest number of layers) at layer l. We combine the neurons that are common till layer $(l-1)$ using the above approach. For e.g. in Fig. 2(b), DPE-1 ends on Conv3 layer, and DPE-2 ends on ConvK layer. Thus, we combine the models till Conv2 layer. From Conv3 layer, we divide the architecture into two parts, (a) one which ends on Conv3 (of smaller network) to Conv3_1 and other to Conv3_2 (of bigger model). Conv3_1 is termed as a branch ending for DPE-1. Further, Conv3_2 connects to rest of the layers till ConvK as shown in Fig. 2(b). We use the same process for the remaining DPEs. Finally, we connect the network output of each branch to a second level FC layer. Here, we add an FC layer on top of Conv3_1, and ConvK, which helps to learn semantic relations between the parts captured at Conv3_1 and ConvK. We do not need a first level FC layer in this case, as these DPE's are part of the same models.

2.5. Modifying N Class Model to N+1 Class Model

We combine the above two proposed architectures to train an existing model to learn a new class through its DPEs. Given a trained model M_i, we want to add new class c whose DPEs extracted from other models are P_i^{js}, where $s \in \{1, .., z\}$, z is the total number of DPEs identified for the class c, the DPEs which come from same architecture are combined using the ShaDPE, and the DPEs which come from different architectures are combined using the SeqDPE as discussed in Sections. 2.3 and 2.4. Overall, the layers of model M_i are combined to the DPEs in a sequential way (See Fig. 2(c)).

To combine the network architectures from the original model and the DPEs, we propose the following two methods- (a) Reduced FC (R-FC) and (b) Complete FC (C-FC) Layers. For R-FC layers, we do not add first level FC layer with M_i, as this is an already fully trained model. We add first level FC layers to each of the DPE architectures. The last layer of the original model, and newly added first level FC layers of DPEs are finally connected to second level FC layer (FC_Final as shown in Fig. 2(c)). For C-FC Layers we add a single FC layer with the model M_i with that of the DPEs. Finally, all the single level FC layers are connected to a second level FC layer.

2.6. Part Neuron(s) Identification for DPE Creation

We identify the part neurons by visually glancing the filters of different layers. For example in Fig. 3 (a), "Wheel" part is represented by three neurons (203, 328, 364) at Conv4 layer, similarly "Headlight" part is represented by neuron 340 at Conv3 layer. In future, our aim to automatically extract DPEs utilizing techniques like DPM [17], Latent SVM [18].

3. RESULTS AND EVALUATION

3.1. Dataset Details

Bus Dataset: We gathered a total of 1500 Bus Class samples (positive class) and randomly sampled 2000 images that did not contain a bus (negative class) from ImageNet [19], Pascal

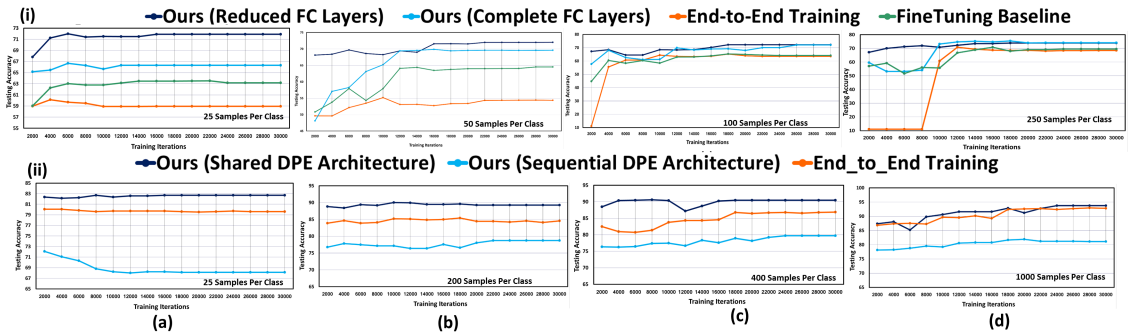

Fig. 4: Plot of test accuracy against the training iterations for varying samples per class on (i) Bus Class Dataset, and (ii) Combined dataset (Caltech-8 + Bus Class)

VOC 2007 [20] and Caltech-256 [21] datasets.

Caltech-8 Dataset: We selected those classes from Caltech-256 dataset [21] where total number of samples for a class was greater than 250 (Billiards, Hammock, Horse, Ladder, Motorbikes, T-Shirt, Airplanes and Face-Easy). Data for these 8 classes constitute the Caltech-8 dataset.

3.2. Baselines

Fine Tuning (FT) Approach: Due to small dataset size, we freeze the first few layers (copied from pre-trained base deep model) and perform training only on the additional new layers.
End-to-End (ETE) Training: A new model is trained from scratch by randomly initializing network weights and requires significant training data to accurately learn network weights.

3.3. Experimental Settings

We evaluate our framework for two tasks (i) Learning Bus Class with SeqDPE and ShaDPE, (ii) Modifying existing deep model to learn Bus Class. Using the concepts discussed in Sec. 2.2, we find the representative neurons of similar looking parts from "Car" and "Train" class (see Fig. 3(a)) and extract corresponding 5 DPEs (see Fig. 3(b)). Finally, we learn these 5 DPEs using proposed SeqDPE and ShaDPE approaches. For Testing of Bus dataset, we used 500 samples from bus class, and 500 samples from negative class. For the task of learning a new class for an existing model, we report the performance on 700 test samples for the 9 classes (Caltech-8 + Bus).

3.4. Discussion

We showcase the effectiveness of proposed ShaDPE and SeqDPE on Bus Class dataset. Fig. 4 shows the plot of test accuracy against the training iteration for different methods and varying amounts of training data samples per class (25, 200, 400, 1000). As seen in Fig. 4 (ii), proposed *ShaDPE* is performing consistently better than *SeqDPE* and significantly better than the baseline even for few samples per class (Fig. 4(a-d)) due to fewer parameters that need to be learned and is able to achieve a reasonably high performance even with as few as 25 samples per class.

For the task of adding a new class to an existing trained deep learning model, we sequentially combine *ShaDPE* for the Bus class with existing trained model on the Caltech-8 dataset and train the complete network with the data for 9 classes (Caltech-8 + Bus Class) with reduced as well as complete FC approach. Fig. 4 (i) shows the comparison of the two networks, with FT and ETE training baselines on 9 classes.

From Fig. 4, we observe that proposed approaches perform higher than both baselines even with few training samples per class (25, 50) Overall, the reduced FC approach shows improved performance with fewer training samples per class and converges with few training iterations. This is because it only needs to retrain the FC layer for the DPE architecture since the FC layer of initial Caltech-8 model is already trained and directly contributes to the softmax output.

Table 1 shows the class-wise performance of the different network architectures for the 9 classes (Caltech-8 + Bus Class). We fine-tune the FC layers of the Caltech-8 model with 25 samples per class which forms the final baseline. We observe that both *ShaDPE* and *SeqDPE* do not significantly affect the performance of the base Caltech-8 model while learning a new class. Also, the networks are able to achieve a reasonably high performance on the new Bus class (74-77%) while the baselines have poor accuracy due to insufficient training data. From these results, we conclude that it is possible to train a deep model over a new class with limited samples given the DPEs for the new class.

4. CONCLUSION

In this paper, we proposed a novel framework for training CNNs with DPEs. We present different ways of combining these part embeddings into a network to learn a class with few training samples. To the best of our knowledge, we present the first attempt for DPE creation and using at for training a deep neural network for different classes and show the effectiveness of our framework against standard baselines.

5. REFERENCES

[1] Maxime Oquab, Leon Bottou, Ivan Laptev, and Josef Sivic, "Learning and transferring mid-level image representations using convolutional neural networks," in *(CVPR)*. IEEE, 2014, pp. 1717–1724.

[2] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell, "Decaf: A deep convolutional activation feature for generic visual recognition," in *ICML*, 2014, pp. 647–655.

[3] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson, "Cnn features off-the-shelf: an astounding baseline for recognition," in *CVPRW*. IEEE, 2014, pp. 512–519.

[4] Joseph J Lim, Ruslan R Salakhutdinov, and Antonio Torralba, "Transfer learning by borrowing examples for multiclass object detection," in *Advances in neural information processing systems*, 2011, pp. 118–126.

[5] Luo Jie, Tatiana Tommasi, and Barbara Caputo, "Multiclass transfer learning from unconstrained priors," in *ICCV*. IEEE, 2011, pp. 1863–1870.

[6] Tianshi Gao, Michael Stark, and Daphne Koller, "What makes a good detector?–structured priors for learning from few examples," in *ECCV*. Springer, 2012, pp. 354–367.

[7] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean, "Distilling the knowledge in a neural network," *arXiv*, 2015.

[8] Tommaso Furlanello, Zachary C Lipton, AI Amazon, Laurent Itti, and Anima Anandkumar, "Born again neural networks," in *NIPS Workshop on Meta Learning*, 2017.

[9] Qiangui Huang, Kevin Zhou, Suya You, and Ulrich Neumann, "Learning to prune filters in convolutional neural networks," *arXiv*, 2018.

[10] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf, "Pruning filters for efficient convnets," *arXiv*, 2016.

[11] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter, "Continual lifelong learning with neural networks: A review," *arXiv*, 2018.

[12] Daniel L Silver, Qiang Yang, and Lianghao Li, "Lifelong machine learning systems: Beyond learning algorithms.," in *AAAI Spring Symposium: Lifelong Machine Learning*, 2013.

[13] Tianqi Chen, Ian Goodfellow, and Jonathon Shlens, "Net2net: Accelerating learning via knowledge transfer," *arXiv*, 2015.

[14] Matthew D Zeiler and Rob Fergus, "Visualizing and understanding convolutional networks," in *ECCV*. Springer, 2014, pp. 818–833.

[15] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson, "Understanding neural networks through deep visualization," *arXiv*, 2015.

[16] "Visualize cnn features in a hierarchy," `https://github.com/mcogswell/cnn_treevis`.

[17] Pedro F Felzenszwalb, Ross B Girshick, David McAllester, and Deva Ramanan, "Object detection with discriminatively trained part-based models," *IEEE transactions on pattern analysis and machine intelligence*, vol. 32, no. 9, pp. 1627–1645, 2010.

[18] Weilong Yang, Yang Wang, Arash Vahdat, and Greg Mori, "Kernel latent svm for visual recognition," in *Advances in neural information processing systems*, 2012.

[19] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR*, 2009.

[20] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *IJCV*, 2010.

[21] Gregory Griffin, Alex Holub, and Pietro Perona, "Caltech-256 object category dataset," 2007.