International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-9 Issue-7, May 2020

Imageem: Pre-Trained Encoded Vector Embeddings for Image Modelling

Siddhartha Dhar Choudhury, Kunal Mehrotra, D Vanusha

Abstract: ***Today some of the most intriguing challenges posed in the field of AI, is the lack of computational resources required to train the deep learning models, especially the image related problems. These Image modelling problems are solved by training a Convolutional Neural Network (CNN) which is computationally expensive process. Our aim is to extract the features of the images by training on these vector embeddings locally and then deploying it in the server for easy access to the researchers. The proposed system provides high dimensional vectors that capture dense features of objects. Pre-trained word embedding is a common way of representing words in a vocabulary present in a document. These embeddings have the capability to capture the context of a particular word in a sentence or an entire document, in relation with words other than the one under review. A popular algorithm for training these word embeddings is skip-gram model proposed in Word2Vec architecture.By using these trained vector embeddings of popular image models, researchers can download them and apply it to their use case which will reduce their training time by a thousand fold. Our aim is to make it so simple for the user that he can train it on edge devices like raspberry pi, jetson nano, these pre-trained vectors image models of high accuracy can be trained on even mobile phone processor.***

Keywords: Neural networks, Model deployment, Edge AI, Deep learning

I. INTRODUCTION

In current deep learning based image modelling systems, Convolutional Neural Network plays a major role. These networks are responsible for extracting important features from images using various filters which are learnt during training with the help of learning algorithms like Gradient Descent or its variants.

These Image modelling problems are solved by training a Convolutional Neural Network (CNN) which is computationally expensive process. Our aim is to extract the features of the images by training on these vector embeddings locally and then deploying it in the server for easy access to the researchers.

CNN based architectures though give high accuracy but are computationally very expensive and it is extremely difficult to reduce the size of these models. As there is a shift from cloud to deploying on edge, and in some use cases it is just not feasible to deploy the model on the cloud and interact with it using internet, for these cases it is extremely tough to get the model size to an acceptable level of run time.

Revised Manuscript Received on May 30, 2020.
***** Correspondence Author
Siddhartha Dhar Choudhury*, SRM Institute of Science and Technology
Kunal Mehrotra, SRM Institute of Science and Technology
D Vanusha, SRM Institute of Science and Technology

We aim to solve the issue of deploying image models on the edge and completely eliminate the use of Convolutional Layers from the image modelling pipeline. We propose a novel technique called image embeddings to solve the aforementioned problem.

The idea of image embedding is inspired from Natural Language Processing literature, from word embeddings to be more specific. Word embeddings are a dense representation of a word in n-dimensional space, what this essentially means is that each word is expressed in form of a n-dimensional vector containing floating point numbers which hold the context and meaning of that particular word. These embeddings when projected in a n-dimensional space are closer to those with similar meaning.

Borrowing from this, we aim to train high dimensional embedding vectors for individual images (or objects), these vectors will then hold important information packed in a dense vector about the object in context. Once these embeddings are trained users can use these with their datasets.

What one requires then is to take a similarity measure of the image (flattened into a 1-D vector) with the pre-trained embeddings belonging to that class, this will create the dataset which can be fed directly into a Multi-Layered Perceptron (MLP) and without any CNN he/she can train the image model.

The main aim of our research area is to help the deep learning practitioners and researchers by saving their time and money from buying computationally expensive resources like GPU's. What we are targeting is the popular image modelling problems like image classification, object detection, image segmentation.

By using the local GPU resources to train these heavy resource dependent models and uploading them on cloud based services, researchers can use these pre-trained vector embeddings which have learnt the dense feature abstractions from the images. They can just make fine readjustments and tweaks according to their model and their model will be ready to go for deployment.

This will encourage the young enthusiasts to freely research and implement the areas which were only possible by large organizations with a team of researchers. This will help us grow the deep learning community and promote in the contribution to open source platform.

Our aim is to make it so simple for the user that he can train it on edge devices like raspberry pi, jetson nano, these pre-trained vectors image models of high accuracy can be trained on even mobile phone processor.

II. RELATED WORK

2.1 Character-level Convolutional Networks for Text Classification

In this article the text classification is being done by the use of convolutional neural networks at character level on empirical exploration.

Many comparisons are done with models like n-grams, bag of words and TFIDF variants which include many dl models like RNN and ConvNets [4].

2.2 Recent Trends in Deep Learning Based Natural Language Processing

Many models have emerged from the context of natural language processing (NLP) [5].The area which this paper focuses is the major development and breakthrough models which are deployed in various NLP tasks which gives a pathway for their evolution.

2.3 A Survey of Model Compression and Acceleration for Deep Neural Networks

The methods used in this paper [10] are divided into roughly four tasks: low-rank factorisation parameter, pruning and sharing, knowledge distillation and transferred/compact convolutional filters. Many techniques will be described at the beginning like pruning and sharing, the other techniques will be introduced. In every scheme which is being used it provides a deep insight regarding advantages, and drawbacks, performance, related applications, etc. After that we take a walkthrough through methods like dynamic capacity and stochastic depth networks.

2.4 SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and $<$ 0.5MB model size

The main advantage which is being offered by the small scale DNN architectures [11] are: (1) In distributed training these DNN's require less communication. (2) From cloud to an autonomous car lesser bandwidth is required by them. (3) Many limited memory applications like the FPGA's can be used to deploy on them. Thus this small scale architecture is called the SqueezeNet.

2.5 Word Embeddings with Limited Memory

In this paper they have studied on the precise data representation and computation on word embeddings [12]. The results are pointing on the 8-bit fixed point value which are possible to use and train without going through the dependency parsing tasks and loss of performance in word or phrase.

2.6 ProdSumNet: reducing model parameters in deep neural networks via product-of-sums matrix decompositions

The product of sum of linear parameters by decomposing the the linear operations by using a general framework [13] for reducing the trainable parameters of the model. The main advantage it has over its competitors is that the number of trainable parameters is fixed and cannot be changes arbitrarily.

2.7 An Innovative Word Encoding Method For Text Classification Using Convolutional Neural Network

The new method for text classification which is being used here is the independent word encoding method for new language [14]. This step requires minimal steps for preprocessing for converting the model into low level feature dimension from raw text data by using a new technique called binary unique number of word "BUNOW" . It gives a unique ID in a dictionary which is equivalent to a k-dimensional vector having a binary equivalent format.The classification task is then done by giving the giving the output of encoding vector in CNN.

2.8 Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

In this paper we make use of the detection network and introduce a new network which is the Region Proposal Network which is sharing the full-image with it, which helps in giving a cost-free regional proposal. The work [15] which is being done by this RPN network is that it gives the objectless score at all positions and simultaneously products bounds for objects. The high quality region proposals which are generated by the RPNs are used by the Faster R-Cnn for detection.

2.9 GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism

Scaling up profound neural system limit has been known as a powerful methodology to improving model quality for a few diverse AI assignments. In numerous cases, expanding model limit past the memory furthest reaches of a solitary quickening agent has required creating uncommon calculations or foundation. These arrangements are regularly engineering explicit and don't move to different undertakings. To address the requirement for productive and task-autonomous model parallelism, we present GPipe [1], a pipeline parallelism library that permits scaling any system that can be communicated as a grouping of layers. By pipelining distinctive sub-groupings of layers on isolated quickening agents, GPipe gives the adaptability of scaling a wide range of systems to colossal sizes proficiently. In addition, GPipe uses a novel batchsplitting pipelining calculation, bringing about practically straight speedup when a model is apportioned over different quickening agents.

2.10 Long-term Recurrent Convolutional Networks for Visual Recognition and Description

Models dependent on profound convolutional systems have overwhelmed ongoing picture understanding assignments; they examine regardless of whether models which are likewise intermittent, or "transiently profound", are powerful for errands including groupings, visual furthermore, something else. They build up a novel intermittent convolutional design appropriate for huge scope visual realizing which is start to finish trainable, and show the estimation of these models on benchmark video acknowledgment assignments, picture portrayal what's more, recovery issues, and video portrayal challenges. As opposed to current models which accept a fixed spatio-worldly open field or straightforward fleeting averaging for consecutive handling, repetitive convolutional models are "doubly profound" in that they can be compositional in spatial and worldly "layers". Such models may have points of interest when target ideas are mind boggling as well as preparing information are restricted. Learning long haul conditions [2] is conceivable when nonlinearities are consolidated into the system state refreshes. Long haul RNN models are engaging in that they legitimately can outline length inputs (e.g., video outlines) to variable length yields (e.g., normal language message) and can display complex transient elements; yet they can be streamlined with backpropagation. Our intermittent long haul models are straightforwardly associated with present day visual convnet models and can be mutually prepared to all the while learn transient elements and convolutional perceptual portrayals.

2.11 Word Embeddings and Their Use In Sentence Classification Tasks

This paper has two sections. In the initial segment they examine word embeddings [3]. They examine the requirement for them, a few of the strategies to make them, and a portion of their fascinating properties. They likewise contrast them with picture embeddings and perceive how word inserting and picture installing can be consolidated to perform various errands. In the second part they actualize a convolutional neural system prepared on pre-prepared word vectors.

The system is utilized for a few sentence-level arrangement errands, and accomplishes condition of-craftsmanship (or equivalent) results, showing the extraordinary intensity of pre-trainted word embeddings over arbitrary ones.

2.12 Deep Neural Network Architecture for Chatacter-Level Learning on Short Text

Character-level profound learning for content characterization undertakings [6] empowers models to be prepared with no earlier information on the information or language; be that as it may, an ideal neural system structure for various content spaces is not known and may change. Right now, develop current endeavors to prepare neural systems from character level information by directing a trial examination on neural system structure for content order of short content archives. They prepared and assessed four systems, two comprising of convolutional layers followed by thick layers and two comprising of convolutional layers followed by a LSTM layer. Their trial results show tweets need arrange models good with their short length. Systems discovered viable for other assessment grouping assignments may not create an viable classifier right now, their engineering is illsuited for short occurrences.

2.13 Large scale distributed deep networks

Ongoing work in unaided component learning and profound learning has demonstrated that being ready to prepare huge models can significantly improve execution. Right now, they think about the issue of preparing a profound system with billions of parameters [7] utilizing a huge number of CPU centers. They have built up a product structure called DistBelief that can use processing groups with a large number of machines to train enormous models. Inside this system, we have created two calculations for enormous scope disseminated preparing: (I) Downpour SGD, a nonconcurrent stochastic angle plummet methodology supporting an enormous number of model reproductions, and (ii) Sandblaster, a structure that bolsters an assortment of disseminated cluster enhancement strategies, including a dispersed execution of L-BFGS. Deluge SGD what's more, Sandblaster L-BFGS both increment the scale and speed of profound system preparing. They have effectively utilized their framework to prepare a profound system 30x bigger than recently announced in the writing, and accomplishes best in class execution on ImageNet, a visual article acknowledgment task with 16 million pictures and 21k classifications. They show that these equivalent systems significantly quicken the preparation of an all the more unobtrusively measured profound system for a business discourse acknowledgment administration. In spite of the fact that they center around and report execution of these techniques as applied to preparing enormous neural systems, the fundamental calculations are relevant to any slope based AI calculation.

2.14 Distributed learning of deep neural network over multiple agents

In spaces, for example, medicinal services and nance, deficiency of named information and computational assets is a basic issue while creating AI calculations. To address the issue of named information shortage in preparing and arrangement of neural system based frameworks [8], they propose another procedure to train profound neural systems more than a few information sources. Their strategy considers profound neural systems to be prepared utilizing information from various elements in a disseminated manner. They assess our calculation on existing datasets and show that it acquires execution which is like an ordinary neural system prepared on a solitary machine. They further extend it to join semi-administered realizing when preparing with hardly any named tests, and investigate any security worries that may emerge. Their calculation makes ready for circulated preparing of profound neural systems in information delicate applications when crude information may not be shared legitimately.

2.15 Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis

Deep Neural Networks (DNNs) are turning into a significant apparatus in present day processing applications. Quickening their preparation is a significant test and methods run from dispersed calculations to low-level circuit structure. Right now, depict the issue from a hypothetical point of view, trailed by approaches for its parallelization [9]. They present patterns in DNN structures and the subsequent ramifications on parallelization procedures. They at that point audit and model the various sorts of simultaneousness in DNNs: from the single administrator, through parallelism in arrange induction and preparing, to appropriated profound learning. They examine nonconcurrent stochastic streamlining, circulated framework designs, correspondence plans, and neural engineering search. In light of those methodologies, we extrapolate potential headings for parallelism in profound learning.

III. ALGORITHMS USED

3.1 Convolutional Neural Networks

This network is specially used and designed to handle pixel data, it is kind of an artificial neural network which has applications in filed like image processing and recognition. The make of CNN includes many layers which are the input layer, output layer and a multilayer perceptron layer included as the hidden layer, then we have the fully connected layers and normal layers.

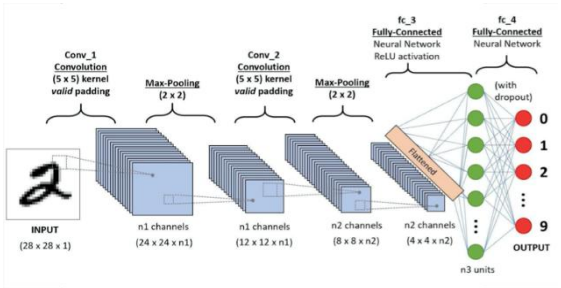

Fig 1: Convolutional Neural Network

3.2 Autoencoder

This type of neural network is used to learn the data encodings in an efficient manner by an artificial neural network in an unsupervised way. What the auto encoder aims to learn is the encoding or the representation of the data set, which is typically done for the task of dimensionality reduction. What is being done is that we are ignoring the signal of the network which is the outlier noise.

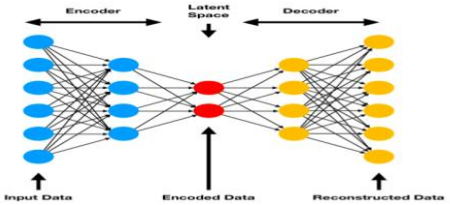

Fig 2: Autoencoder

3.3 Multi Layered Perceptron

A logistic regressor needs an input which is done by inserting the intermediate layer instead, this layer is called the multilayer perceptron and the intermediate layer is called the hidden layer, which has a non linear activation function which can be sigmoid or tanh. The researchers thus can use many layers making their model architecture very big and deep.

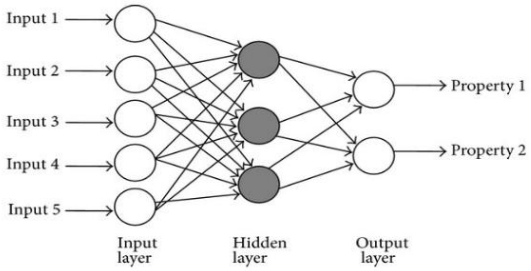

Fig 3: Multi Layered Perceptron

3.4 Vector Products

Vector, be it in Machine Learning or Linear Algebra refers to the same - a collection / array of numbers - example: [1,3,2] is a vector. In machine learning this vector is called a feature vector as each of these values corresponds to some features, say features of a fruit in a fruit classification problem.

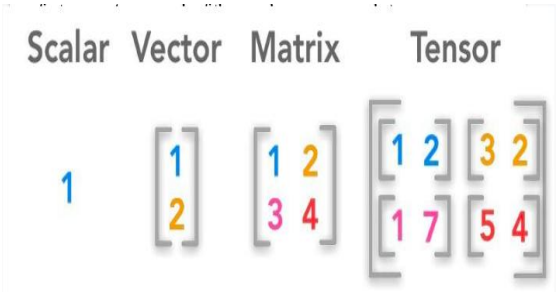

Fig 4: Vector Products

IV. UNDERSTADNING IMAGEEM

ImageEm is a set of pre-trained dense vector representations of various daily life objects which can be used directly for inference without involvement of any training procedure. Mostly image modelling problems use Convolutional Neural Networks for these tasks, but a major down side of these models is the amount of time required for training these models.

The three steps to creation and usage of ImageEm embeddings are:-

4.1 Data Scraping

The first module is used to collect data for training the image models. This is done by scraping images of a particular object from Google Images using selenium web driver for Python. As, google images has images of a wide array of objects, and also allows for free usage in training models, we chose to fetch the data from google images. Also, google image results vary depending on the person from whose account it is being accessed so from both of our accounts we get around 600-700 images for each category.

4.2 Training the high dimensional image embedding vectors

The embedding vectors are trained using a Convolutional Neural Network based architecture called an Auto Encoder. An auto encoder has two parts - an encoder and a decoder.

The job of the encoder is to encode an image into a set size dense vector. This is done using successive convolution operations, this process is called down sampling of an image and maintains only the most important features of the images and loses unnecessary features in the process.

The second part is called the decoder, this takes the dense vectors as inputs and generates the image from this using the information from the floating point numbers in the vector. This is done using deconvolution operation in a process called up sampling.

The final output of this network is the same image as the input. Once the output image is very close to the input image we stop the training, after this we can say that the dense vectors at the middle of the network (the output of the encoder and the input of the decoder) is a dense representation of the object in the input image. These values are then stored in a text file.

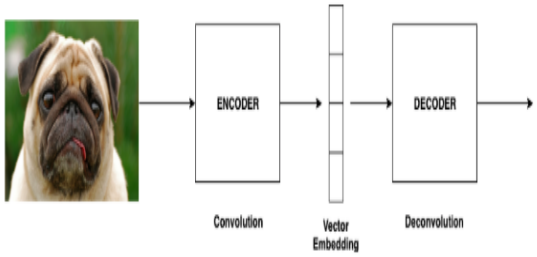

Fig 5: Training image embeddings using Autoencoders

4.3 Training an image classifier using pre-trained embeddings

The image is to be converted into 1-D vector by flattening it row-wise. Similarity index between the image and the pre-trained image embedding. This is then treated as the dataset for the image model. This will convert the image input into a vector of floating point numbers. This problem can now be handled using a Multi-Layered Perceptron (MLP). So, training an image model is much less computationally expensive than training a Convolutional Neural Network. This image model is an alternative to Convolutional Neural Networks based image models. The models trained using ImageEm are as accurate as those trained using CNN and at the same time these are computationally very inexpensive and thus can be trained as well as deployed on edge devices such as Raspberry Pi, Mobile Phones and other small microcontrollers. There is also little to no compromise when it comes to accuracy of the trained model and we have achieved state of the art results with these pre-trained networks.

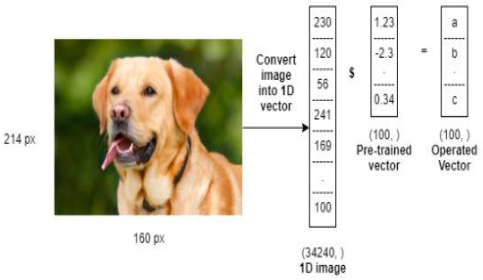

Fig 6: Training image classifier using pre-trained embeddings

V. EXPERIMENTATION

The following experiments were conducted to verify the ImageEm pre-trained vector embeddings:

5.1 Binary classification

A convolutional neural network was trained for binary classification for differentiating between dog and cat images.

The dataset contains 10,000 (ten thousand) images of different breeds of dogs and cats, with 5,000 images of each type. The entire dataset was split into three parts - 6,000 images for training autoencoders to obtain dog and cat high dimensional vector embedding (ImageEm), 1,000 images for usage as validation set during training of image embeddings, 2,000 images for testing the sanity of the trained vector embeddings (these are the images which will be used by a deep learning practitioner for getting a prediction of class of object in an image).The autoencoder has two parts - an encoder and a decoder. The encoder takes as input an image and applies convolution and max pooling operations on the image to transform it into a final one dimensional vector representation. The decoder takes in as input the vectorized representation of the image containing the important features of the input image which were extracted by the encoder network. This one dimensional vector is then passed through deconvolution or up sampling layers in the decoder that converts the 1D vector into an image. The final aim of training an autoencoder is to get the same image as output as was given in the output, thus we can say that the vector given as output by the encoder is a dense representation of the type of object passed into the network. From the 6,000 images set apart for training the autoencoders, the data was split into two parts of 3,000 images each. Each part contains images belonging to a single class (dog and cat). Two autoencoders are trained each for one class of our classification model. The details of the two autoencoders are given below:

- **Autoencoder 1 (trained on dog images)**
  - **Number of layers in encoder**: 4
  - **Number of filters used in each layer in encoder**: 32, 64, 64, 64
  - **Size of kernel in encoder layers**: 3, 5, 5, 5 (values are given for respective layers)
  - **Activation function used in encoder (till 3rd layer)**: Rectified Linear Unit (ReLU)
  - **Activation function used in encoder (last/4th layer)**: Linear Activation–Dimension of vectorized representation: 1,000 (1000-D)
  - **Number of layers in decoder**: 4
  - **Number of filters used in each layer in decoder**: 64, 64, 64, 32
  - **Size of kernel in decoder layers**: 3, 5, 5, 5 (values are given for respective layers)
  - **Activation function used in decoder (till 3rd layer)**: Leaky Rectified Linear Unit (Leaky ReLU)
  - **Activation function used in decoder (last/4th layer)**: Linear Activation
- Autoencoder 2 (trained on cat images)
  - **Number of layers in encoder**: 3–Number of filters used in each layer in encoder: 32, 64, 64
  - **Size of kernel in encoder layers**: 3, 5, 5 (values are given for respective layers)
  - **Activation function used in encoder (till 2nd layer)**: Rectified Linear Unit (ReLU)
  - **Activation function used in encoder (last/3rd layer)**: Linear Activation–Dimension of vectorized representation: 1,000 (1000-D)
  - **Number of layers in decoder**: 3
  - **Number of filters used in each layer in decoder**: 64, 64, 32
  - **Size of kernel in decoder layers**: 3, 5, 5 (values are given for respective layers)
  - **Activation function used in decoder (till 2nd layer)**: Leaky Rectified Linear Unit (Leaky ReLU)
  - **Activation function used in decoder (last/3rd layer)**: Linear Activation

The following are results of the experiment:

- **Accuracy on 2,000 images when trained using a Vanilla CNN**= 83%
- **Accuracy on 2,000 images when inferred using ImageEm embeddings**= 88.5%
- **Time required for a single inference using trained CNN**=0.13 seconds
- **Time required for a single inference using ImageEm embed-dings**= 0.034 seconds

5.2 Multi-class classification

In multi-class classification the final dense layer of the entire Convolutional Neural Network has softmax activation function applied to each of the nodes present. This converts the final value of each nodes into a probability, and adding the values of all the nodes for a single instance of inference gives a sum of approximately 1 (which denotes the nature of the value stored in these outputs as being part of a probability distribution).

This multi-class classifier requires a lot of training data to learn features of individual classes and is thus a computationally expensive task. At times, deep learning practitioners or scientists do not have access to such large databases to train their models on. To solve this issue ImageEm pre-trained vector embeddings can be used which have learnt dense feature vectors for objects belonging to multiple classes and can thus be used to directly infer the predicted class given an image without requiring any additional training.

Each ImageEm embedding has a set of dense learnt features as they are trained on huge and sane databases of multiple objects. The training procedure of these vector embeddings are specified in the earlier experiments well as section 4. These image embeddings thus help us to convert the images into a similarity index and then classified into proper class. There were 50,000 images for training belonging to 5 classes and the validation set had 10,0000 images during training the object embeddings (ImageEm). For testing the trained vector embeddings we had 3,000 images belonging to the 5 classes, we trained a Convolutional Neural Network on it and tested the results in parallel with ImageEm predictions. The details of model architecture are given below:

- Autoencoder (a common autoencoder architecture for training 10 different image embeddings)
  - **Number of layers in encoder**: 10
  - **Number of filters used in each layer in encoder**: 64, 64, 64, 64,64, 64, 64, 64, 128, 128
  - **Size of kernel in encoder layers**: 3, 3, 3, 3, 3, 3, 3, 3, 3, 5, 5(values are given for respective layers)–Activation function used in encoder (till 9rd layer): Rectified Linear Unit (ReLU)
  - **Activation function used in encoder (last/10th layer)**: Linear Activation
  - **Dimension of vectorized representation**: 5,000 (5000-D)
  - **Number of layers in decoder**: 10
  - **Number of filters used in each layer in decoder**: 128, 128, 128,128, 128, 128, 128, 128, 64, 64
  - **Size of kernel in decoder layers**: 5, 5, 5, 5, 5, 5, 5, 5, 3, 3(values are given for respective layers)
  - **Activation function used in decoder (till 9th layer)**: Leaky Rectified Linear Unit (Leaky ReLU)
  - **Activation function used in decoder (last/10th layer)**: Linear Activation

The following are results of the experiment:

- **Accuracy on 3,000 images when trained using a Vanilla CNN**= 73.4%
- **Accuracy on 3,000 images when inferred using ImageEm embeddings**= 82.42%
- **Time required for a single inference using trained CNN**=1.2 seconds
- **Time required for a single inference using ImageEm embeddings**= 0.68 seconds

VI. CONCLUSION

From the experimentations it is evident that ImageEm pre-trained encoded image embeddings not only increase the accuracy during inference on test images but also performs the prediction step at a much faster rate as com-pared to Vanilla Convolutional Neural Networks (Vanilla CNNs). Since, ImageEm embeddings are trained on large databases of particular object of interest, the features extracted in these embeddings are a near perfect representation of the images in the testing set. On the other hand, a normal CNN trained on the same dataset gives lesser accuracy due to its failure in capturing some important features from the image. So, usage of Image Embeddings leads to efficient inference which can even be performed on an edge device like Raspberry Pi without even requiring training on newer images.

We would like to extend this work in future to apply ImageEm techniques to other image modelling techniques such as - Object detection, Image segmentation, Generative adversarial networks, etc. We would also like to compare our results with the state-of-the-art algorithms available for these problems.

ACKNOWLEDGEMENT

It is an honour to acknowledge Geoffrey Hinton, Andrew Ng, Yoshua Bengio for their deep learning courses in Udacity, Coursera, and YouTube which was a starting point of our exploration of deep learning and Artificial Intelligence in general. We would also like to give a hearty thanks to Shanky Pandey for proofreading, providing important inputs to the project, and invigorating our innovation. At the end we would extend our acknowledgements to Rusheel Gupta for his support in LATEX typesetting.

REFERENCES

1. Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Mia XuChen, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V. Le,Yonghui Wu, Zhifeng Chen, “GPipe: Easy Scaling with Micro-Batch Pipeline Parallelism”, CoRR, 2018.
2. Jeff Donahue, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko, Trevor Darrell, ”Long-term Recurrent Convolutional Networks for Visual Recognition and Description,” IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 39 , Issue: 4 , April 1 2017).
3. Amit Mandelbaum, Adi Shalev,” Word Embeddings and Their Use In Sentence Classification Tasks,” arXiv preprint arXiv:1610.08229 (2016).
4. Xiang Zhang, Junbo Zhao, Yann LeCun,” Character-level convolutional networks for text classification,” Proceeding NIPS’15 Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1.
5. Tom Young, Devamanyu Hazarika, Soujanya Poria, Erik Cam-bria,”Recent Trends in Deep Learning Based Natural Language Pro-cessing,” IEEE Computational Intelligence Magazine (Volume: 13 ,Issue: 3 , Aug. 2018).
6. Joseph D. Prusa, Taghi M. Khoshgoftaar, ” Deep Neural Network Architecture for Character-Level Learning on Short Text,” Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference.
7. Jeffrey Dean, Greg S. Corrado, Rajat Monga, Kai Chen, MatthieuDevin, Quoc V. Le, Mark Z. Mao, Marc’Aurelio Ranzato, Andrew Senior, Paul Tucker, Ke Yang, Andrew Y. Ng,” Large scale distributed deep networks,” Proceeding NIPS’12 Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume1.
8. Otkrist Gupta, Ramesh Raskar, ” Distributed learning of deep neural network over multiple agents,” J. Network and Computer Applications 2018.
9. Tal Ben-Nun and Torsten Hoefler, ”Demystifying Parallel and Distributed Deep Learning: An In-Depth Concurrency Analysis, ” ACM Computing Surveys (CSUR) Surveys Homepage archive Volume 52 Is-sue 4, September 2019 Article No. 65.
10. Yu Cheng, Duo Wang, Pan Zhou, Member, IEEE, and Tao Zhang, Senior Member, IEEE, ”A Survey of Model Compression and Acceleration for Deep Neural Networks,” arXiv preprint arXiv:1710.09282.
11. Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf,William J. Dally, Kurt Keutzer, ”SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and ¡0.5MB model size,” arXiv preprintarXiv:1602.07360.
12. Shaoshi Ling, Yangqiu Song and Dan Roth, ”Word Embeddings with Limited Memory,” Proceedings of the 54th Annual Meeting of the As-sociation for Computational Linguistics (Volume 2: Short Papers).
13. Chai Wah Wu, ”ProdSumNet: reducing model parameters in deep neural networks via product-of-sums matrix decompositions,” arXiv preprint arXiv:1809.02209.
14. Amr Adel Helmy, Yasser M. K. Omar, Rania Hodhod, ” An Innovative Word Encoding Method For Text Classification Using Convolutional Neural Network,” 14th International Computer Engineering Conference(ICENCO).
15. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun, ”Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” IEEE Transactions on Pattern Analysis and Machine Intelligence (Volume: 39 , Issue: 6 , June 1 2017).