Materialization Trade-offs for Feature Transfer from Deep CNNs for Multimodal Data Analytics

Supun Nakandala Arun Kumar
University of California, San Diego
{snakanda,arunkk}@eng.ucsd.edu

1 INTRODUCTION & MOTIVATION

Deep convolutional neural networks (CNNs) have revolutionized computer vision, yielding state-of-the-art accuracy for many image understanding tasks [17]. The main technical reason for their success is how they extract a hierarchy of relevant parametrized features from raw images, with the parameters learned automatically during training [13]. Each layer of a deep CNN learns a different level of abstraction in terms of what the features capture, e.g., low-level edges and patterns in the lowest layers all the way to abstract object shapes in the highest layers. This remarkable ability of deep CNNs is illustrated in Figure 1.

The success of deep CNNs presents an exciting opportunity to holistically integrate image data into traditional data analytics applications in the enterprise, Web, healthcare, and other domains that have hitherto relied mainly on structured data features but had auxiliary images that were not exploited. For instance, product recommendation systems such as Amazon are powered by ML algorithms that relied mainly on structured data features such as price, vendor, purchase history, etc. Such applications are increasingly using deep CNNs to exploit product images by extracting visually-relevant features to help improve ML accuracy, especially for products such as clothing and footwear [18].

Since training deep CNNs from scratch is expensive in terms of both resource costs (e.g., one might need many GPUs [1]) and the number of labeled examples needed, an increasingly popular paradigm to handle image data is *transfer learning* [19]. Essentially, one uses a pre-trained deep CNN, e.g., ImageNet-trained AlexNet [11, 15] and reads off a certain layer of the features it produces on an image as the image's representation [7, 12]. Any downstream ML model can now operate on these image features along with the structured features. Thus, such *feature transfer* helps reduce costs dramatically for using deep CNNs.

Alas, feature transfer creates a new bottleneck for data scientists in practice: it is impossible to say in general which layer of a CNN will yield the best accuracy for the downstream ML task [10]. The common guideline is to extract and compare multiple layers of CNN features [10, 21]. This is a *model selection* process that combines CNN features and structured data [16]. Perhaps surprisingly, the current dominant approach to handling feature transfer at scale is for data scientists to manually *materialize* each CNN layer from scratch as flat files using tools such as TensorFlow [8], load such data into a scalable data analytics system for downstream ML tasks. Apart from reducing the productivity of data scientists, such manual management of feature transfer workloads leads to wasted opportunities to reuse and optimize computations, which raises runtimes and in turn, costs, especially in the cloud.

In this work, we aim to resolve the above issues for large-scale feature transfer with deep CNNs for multimodal data analytics. We

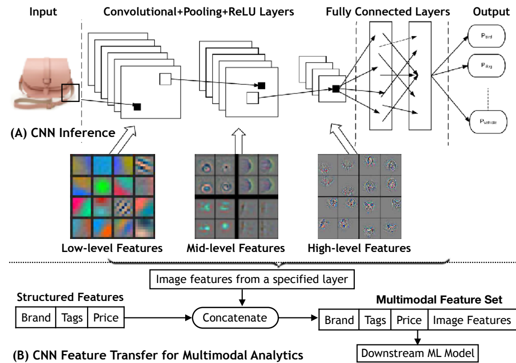

Figure 1: (A) Visualization of CNN features (based on [22]) and (B) Multimodal analytics which uses CNN features.

start with a simple but crucial observation: the different layers of a typical CNN are not independent–*extracting a higher layer requires a superset of the computations needed for a lower layer.* Thus, instead of materializing all layers from scratch, we can reuse previously created layers, subject to other system constraints such as memory or storage. This is a novel instance of a classical database systems-style concern: *materialization trade-offs.*

One might argue: *Why not materialize and cache all layers of interest in one go and use a layer as needed?* While this avoid redundant computations, it increases *memory pressure*, since CNN features are often orders of magnitude larger than the input. Such data blowup lead to non-trivial systems trade-offs for handling memory usage at scale. In fact, performed naively, it could cause system crashes, which would frustrate data scientists and raise costs by forcing them to manually tweak the system or use needlessly more expensive machines. Thus, overall, large-scale feature transfer is technically challenging due to two key systems-oriented concerns: *efficiency* (reducing runtimes) and *reliability* (avoiding system crashes).

To the best of our knowledge, ours is the first work to formalize and study the materialization trade-offs of the emerging workload of large-scale feature transfer from deep CNNs for multimodal analytics over image and structured data from a systems standpoint. We devise a novel optimizer to handle such trade-offs automatically and build a system (named Vista) on top of Spark-TensorFlow combine to enable data scientists to focus on their ML exploration instead of being bogged down by systems issues.

2 PROBLEM STATEMENT

As the input we take two tables $T_{str}(\underline{ID}, X)$ and $T_{img}(\underline{ID}, I)$, where ID is the primary key (identifier), $X \in \mathbb{R}^{d_s}$ is the structured feature vector (with d_s features, including label), and I are raw images. We

are also given a CNN f with n_l layers, a set of layer indices $L \subset [n_l]$ specific to f that are of interest for transfer learning, a downstream ML algorithm M (e.g., logistic regression), a set of system resources R (number of cores, system memory, and number of nodes). The feature transfer workload is to train M for each of the $|L|$ feature vectors obtained by concatenating X with the respective feature layers obtained by partial CNN inference. More precisely, we can state the the workload using the following set of logical queries:

$$\forall\, l \in L : \tag{1}$$

$$T'_{img,l}(\underline{ID}, \hat{f}_l(I)) \;\leftarrow l^{th} \text{ layer CNN features from images} \tag{2}$$

$$T'_l(\underline{ID}, X'_l) \;\leftarrow\; T_{str} \bowtie T'_{img,l} \tag{3}$$

$$\text{Train } M \text{ on } T'_l \text{ with } X'_l \equiv [X, \hat{f}_l(I)] \tag{4}$$

Step (2) performs partial CNN inference to materialize feature layer l. Step (3) concatenates structured and image features using a key-key join. Step (4) trains M on the new multimodal feature vector. The current dominant practice is to run the above queries as such, i.e., materialize all feature layers *manually* and *independently* as flat files and transfer them. Apart from being cumbersome, such an approach is inefficient due to *redundant* partial CNN inference and/or runs the risk of system crashes due to poor memory management. Our goal is to resolve these issues. *Our approach is to elevate this workload to a declarative level, obviate manual feature transfer, automatically reuse partial CNN inference results, and optimize the system configuration and execution for better reliability and efficiency.*

3 SYSTEM OVERVIEW

We prototype Vista as a library on top of the Spark-TensorFlow combine [2, 6]. Figure 2 illustrates our system's architecture. It has four main components: (1) a "declarative" API, (2) a roster of popular named deep CNNs with named feature layers (we currently support AlexNet [15], VGG16 [20], and ResNet50 [14]), (3) the Vista optimizer, and (4) connectors to Spark and TensorFlow (TF). The declarative front-end API is implemented in Python; a user should specify four inputs. First is the system environment (memory, number of cores and nodes). Second is the deep CNN f and the feature layers L (from the roster) to explore for transfer learning. Third are the data tables T_{str} and T_{img}. Fourth is the downstream ML routine (with all its parameters)–currently MLlib's logistic regression.

Under the covers, Vista uses the above inputs and invokes its optimizer to obtain a reliable and efficient combination of decisions for the logical execution plan, key system configuration parameters, and physical execution. After configuring Spark accordingly, Vista runs within the Spark Driver process to orchestrate the feature transfer task by issuing a series of queries in Spark's *DataFrame* (or *DataSet*) API [9]. Vista uses the *TensorFrames* API [6] to invoke TF during query execution to execute our user-defined functions for partial CNN inference and to handle image and feature tensors using custom *TensorList* datatype. Vista specifies the computational graphs to be used by TF based on the user's inputs. Finally, Vista invokes MLlib in the manner determined by our optimizer and returns all trained downstream models. *Overall, Vista frees users from having to manually handle TF code, large feature files, RDD joins, or Spark tuning for such feature transfer workloads.*

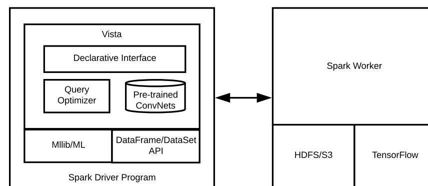

Figure 2: High-level architecture of Vista on top of the Spark-TensorFlow combine.

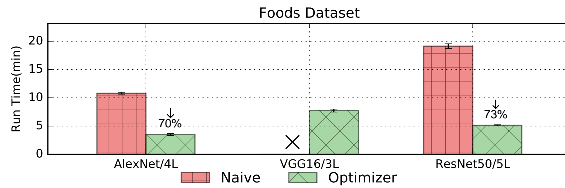

Figure 3: End-to-end reliability and efficiency. "×" indicates a system crashes due to insufficient resources.

4 INITIAL RESULTS & ONGOING WORK

We empirically validate if Vista is able to improve efficiency and reliability of feature transfer workloads by testing it on the *Foods* [4] real world dataset. *Foods* has about 20,000 examples with 130 numeric structured features such as nutrition facts along with pairwise/ternary feature interactions. An image of each food item is given in JPEG format. The target represents if the food is considered healthy. As the layers to select features from, we select *conv5* to *fc8* from AlexNet ($|L| = 4$); *fc6* to *fc8* from VGG ($|L| = 3$), and top 5 layers from ResNet ($|L| = 5$). As for M, we run MLlib's logistic regression for 10 iterations. We compare two approaches: *Naive* and Vista. *Naive* is the current dominant practice of running all feature transfer queries separately, with Spark configured by standard practices [3, 5]. Vista is the plan picked by the Vista optimizer, including for Spark system configuration values such as heap size and number of cores per executor. Figure 3 shows the results.

We see that Vista improves both efficiency and reliability. With *VGG16*, *Naive* simply crashes. This is due to blowups in JVM Native Memory during the CNN inference due to large CNN model footprints, which the Naive approach doesn't account for. However, Vista finishes in reasonable time, which could reduce both user frustration and costs. As for the other cases where *Naive* does not crash, Vista improves efficiency significantly, reducing runtimes by 70%–73%. This reduction arises because Vista removes redundancy in iterative CNN inference and chooses appropriate system parameters.

Currently we are looking into how different logical plan re-writes, physical plan choices and system configuration values will affect the the runtime of CNN feature transfer workloads with varying the data scale, number of layers explored and different CNN models. Ultimately Vista optimizer will evaluate these choices and will pick the best execution plan for the selected CNN feature transfer workload. As future work we are planning to explore several other workloads such as interpretability workloads in CNN feature transfer and optimize them from a systems standpoint.

REFERENCES

[1] Benchmarks for popular cnn models. https://github.com/jcjohnson/cnn-benchmarks. Accessed December 31, 2017.

[2] Deep learning with apache spark and tensorflow. https://databricks.com/blog/2016/01/25/deep-learning-with-apache-spark-and-tensorflow.html. Accessed December 31, 2017.

[3] Distribution of executors, cores and memory for a spark application running in yarn. https://spoddutur.github.io/spark-notes/distribution_of_executors_cores_and_memory_for_spark_application. Accessed December 31, 2017.

[4] Open food facts dataset. https://world.openfoodfacts.org/. Accessed December 31, 2017.

[5] Spark best practices. http://blog.cloudera.com/blog/2015/03/how-to-tune-your-apache-spark-jobs-part-2/. Accessed December 31, 2017.

[6] Tensorframes: Tensorflow wrapper for dataframes on apache spark. https://github.com/databricks/tensorframes. Accessed December 31, 2017.

[7] Transfer learning with cnns for visual recognition. http://cs231n.github.io/transfer-learning/. Accessed December 31, 2017.

[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. Tensorflow: A system for large-scale machine learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283. USENIX Association, 2016.

[9] M. Armbrust, R. S. Xin, C. Lian, Y. Huai, D. Liu, J. K. Bradley, X. Meng, T. Kaftan, M. J. Franklin, A. Ghodsi, et al. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 1383–1394. ACM, 2015.

[10] H. Azizpour, A. S. Razavian, J. Sullivan, A. Maki, and S. Carlsson. Factors of transferability for a generic convnet representation. *IEEE transactions on pattern analysis and machine intelligence*, 38(9):1790–1802, 2016.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[12] J. Donahue, Y. Jia, O. Vinyals, J. Hoffman, N. Zhang, E. Tzeng, and T. Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In E. P. Xing and T. Jebara, editors, *Proceedings of the 31st International Conference on Machine Learning*, volume 32 of *Proceedings of Machine Learning Research*, pages 647–655, Bejing, China, 22–24 Jun 2014. PMLR.

[13] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. The MIT Press, 2016.

[14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[16] A. Kumar, R. McCann, J. Naughton, and J. M. Patel. Model selection management systems: The next frontier of advanced analytics. *ACM SIGMOD Record*, 44(4):17–22, 2016.

[17] Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436–444, 5 2015.

[18] J. McAuley, C. Targett, Q. Shi, and A. Van Den Hengel. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–52. ACM, 2015.

[19] S. J. Pan and Q. Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[20] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[21] J. Yosinski, J. Clune, Y. Bengio, and H. Lipson. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3320–3328. MIT Press, 2014.

[22] M. D. Zeiler and R. Fergus. Visualizing and understanding convolutional networks. In *European conference on computer vision*, pages 818–833. Springer, 2014.