Vista: Declarative Feature Transfer from Deep CNNs at Scale

Addendum

#220

A Estimating Intermediate Data Sizes

We explain the size estimations in the context of Spark. Ignite also uses an internal format similar to the Spark. Spark's internal binary record format is called "Tungsten record format," shown in Figure 1. Fixed size fields (e.g., float) use 8 B. Variable size fields (e.g., arrays) have an 8 B header with 4 B for the offset and 4 B for the length of the data payload. The data payload is stored at the end of the record. An extra bit tracks null values.

Vista estimates the size of intermediate tables $T_l \; \forall l \in L$ based on its knowledge of the CNN. For simplicity, assume *ID* is a long integer and all features are single precision floats. Let $|X|$ denote the number of features in X. $|T_{str}|$ and $|T_{img}|$ are straightforward to calculate, since they are the base tables. For $|T_i|$ with feature layer $l = L[i]$, we have:

$$|T_i| = \alpha_1 \times (8 + 8 + 4 \times |g_l(\hat{f}_l(I))|) + |T_{str}| \quad (1)$$

Equation 1 assumes deserialized format; serialized (and compressed) data will be smaller. But these estimates suffice as safe upper bounds.

Figure 2 shows the estimated and actual sizes. We see that the estimates are accurate for the deserialized in-memory data with a reasonable safety margin. Interestingly, *All-at-a-Time* (AaT) is not that much larger than *Staged* for AlexNet. This is because among its four layers explored the 4^{th} layer from the top is disproportionately large while for the other two layer sizes are more comparable. Serialized is smaller than deserialized as Spark compresses the data. Interestingly, AlexNet feature layers seem more compressible; we verified that its features had many zero values. On average, AlexNet features had only 13.0% non-zero values while VGG16's and ResNet50's had 36.1% and 35.7%, respectively.

B Pre Materializing a Base Layer

Often data scientists are interested in exploring few of the top most layers. Hence, a base layer can be pre-materialized before hand for later use of exploring other layers. This can save computations and thereby reduce the runtime of the CNN feature transfer workload.

However, the CNN feature layer sizes (especially for conv layers) are generally larger than the compressed image formats such as JPEG (see Table 1). This not only increases the secondary storage requirements but also increases the IO

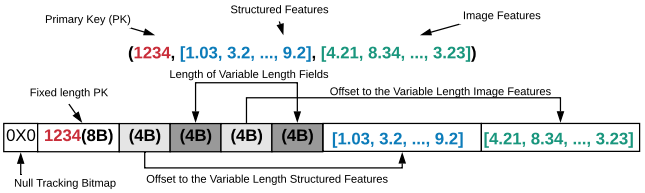

Figure 1. Spark's internal record storage format

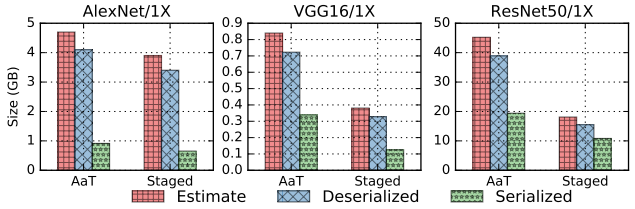

Figure 2. Size of largest intermediate table.

cost of the CNN feature transfer workload both when initially reading data from the disk and during join time when shuffling data over the network.

Table 1. Sizes of pre-materialized feature layers for the **Foods** dataset (size of raw images is 0.26 GB).

	Materialized Layer Size (GB) (layer index starts from the last layer)			
	1^{st}	2^{nd}	4^{th}	5^{th}
AlexNet	0.08	0.14	0.72	
VGG16	0.08	0.20	1.19	
ResNet50	0.08	2.65	3.45	11.51

We perform a set of experiments using the Spark-TF system to explore the effect of pre-materializing a base layer (1, 2, 4, and 5^{th} layers from top). For evaluating the ML model for the base layer no CNN inference is required. But for the other layers partial CNN inference is performed starting from the base layer using the *Staged/After Join/Deserialized/Shuffle* logical-physical plan combination. Experimental set up is same as in Section 5.

For AlexNet and VGG16 when materializing 4^{th}, 2^{nd}, and 1^{st} layers from the top, the materialization time increases as evaluating higher layer requires more computations (see Figure 3 (A) and (B)). However, for ResNet50 there is a sudden drop from the materialization time of 5^{th} layer features to

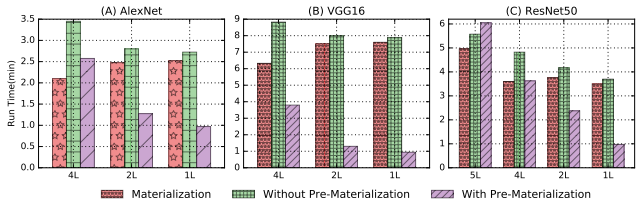

Figure 3. Runtimes comparison for using pre-materialized features from a base layer

the materialization time of 4^{th} layer features. This can be attributed to the high disk IO overhead of writing out 5^{th} layer image features which are ~3 times larger than that of 4^{th} layer (see Figure 3 (C)). Therefore, for ResNet50 starting from a pre-materialized feature layer, instead of raw images, may or may not decrease the overall CNN feature transfer workload runtime.

C Runtime Breakdown

We drill-down into the time breakdowns of the workloads on Spark-TF environment and explore where the bottlenecks occur. In the downstream logistic regression (LR) model, the time spent for training the model on features from a specific layer is dominated by the runtime of the first iteration. In the first iteration partial CNN inference has to be performed starting either from raw images or from the image features from the layer below and the later iterations will be operating on top of the already materialized features. Input read time is dominated by reading images as there are lot of small files compared to the one big structured data file. This is a known issue in HDFS which is also called the "HDFS small file problem." Table 2 summarizes the time breakdown for the CNN feature transfer workload. It can be seen that most of the time is spent on performing the CNN inference and LR 1^{st} iteration on the first layer (e.g 5^{th} layer from top for ResNet50) where the CNN inference has to be performed starting from raw images.

We also separately analyze the speedup behavior for the input image reading and the sum of CNN inference and LR 1^{st} iteration times (see Figure 4). When we separate out the sum of CNN inference and LR 1^{st} iteration times, we see slightly super linear speedups for ResNet50, near linear speedups for VGG16, and slightly better sub-linear speedups for AlexNet.

D Accuracy

For both Foods and a sample of Amazon (20,000 records) datasets we evaluate the downstream logistic regression model F1 score with (1) only using structured features, (2) structured features combined with "Histogram of Oriented Gradients (HOG)" based image features, and (3) structured features combined with CNN based image features from different layers of AlexNet and ResNet models.

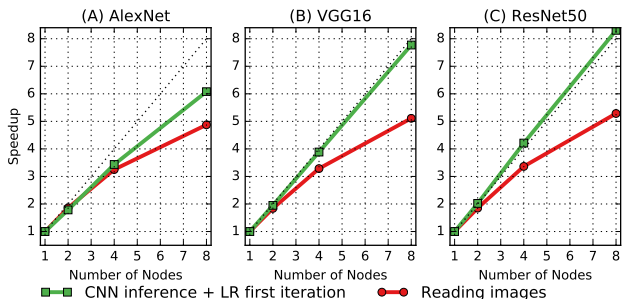

Figure 4. Drill-down analysis of Speedup Curves

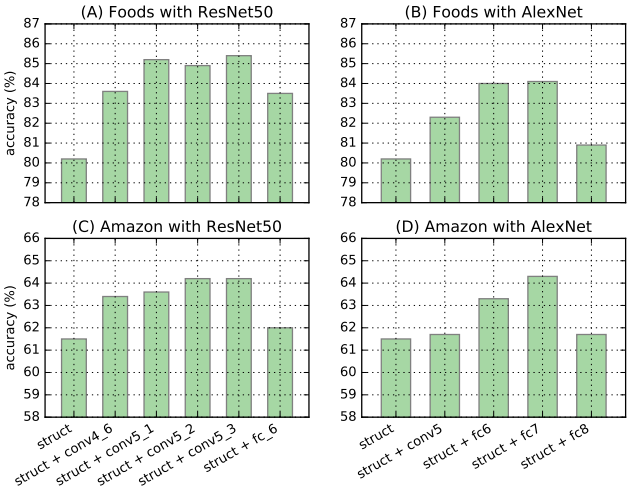

Figure 5. F1 score lifts obtained by incorporating HOG descriptors and CNN features for logistic regression model with elastic net regularization with $\alpha = 0.5$ and a regularization value of 0.01.

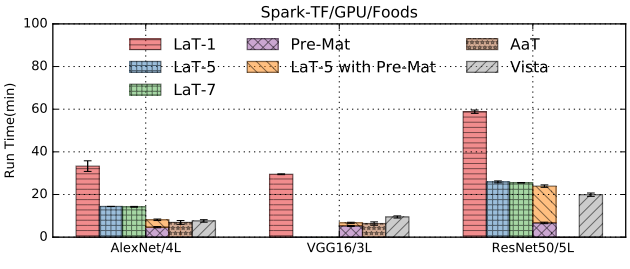

Figure 6. End-to-end reliability and efficiency on GPU. "×" indicates a system crash.

In all cases incorporating image features improves the classification accuracy and the improvement achieved by incorporating CNN features is higher than the improvement achieved by incorporating traditional HOG features (see Table 3 and Figure 5).

Table 2. Runtime breakdown for the image data read time and 1^{st} iteration of the logistic regression model (Layer indices starts from the top and runtimes are in minutes).

		ResNet50/5L				AlexNet/4L				VGG16/3L			
		Number of nodes				Number of nodes				Number of nodes			
		1	2	4	8	1	2	4	8	1	2	4	8
Layer	5	19.0	9.5	4.5	2.3								
	4	3.8	1.8	0.9	0.4	3.7	2.1	1.2	0.7				
	3	2.7	1.3	0.7	0.4	2.4	1.3	0.7	0.5	43.0	22.0	11.0	5.4
	2	2.6	1.3	0.6	0.3	1.1	0.6	0.3	0.2	1.0	0.5	0.3	0.2
	1	1.8	0.9	0.4	0.2	0.3	0.2	0.1	0.1	0.3	0.2	0.1	0.1
	total	29.9	14.8	7.1	3.6	7.5	4.2	2.3	1.5	44.3	22.7	11.4	5.7
Read images		3.7	2.0	1.1	0.7	3.9	2.1	1.2	0.8	4.6	2.5	1.4	0.9

Table 3. F1 scores of test datasets obtained by incorporating HOG descriptors and CNN features for logistic regression model with elastic net regularization with $\alpha = 0.5$ and a regularization value of 0.01.

	Structured Only	Structured + HOG	Structured + CNN
Foods	80.2	81.1	85.4
Amazon	61.5	62.2	64.3

E End-to-End Reliability and Efficiency on GPUs

GPU experiments are run on Spark-TensorFlow environment using the *Foods* dataset. Spark TensorFrames library was modified by adding TensorFlow GPU dependencies to enable GPU support. The experimental setup is a single node machine which has 32 GB RAM, Intel i7-6700 @ 3.40GHz CPU which has 8 cores, 1 TB Seagate ST1000DM010-2EP1 SSD, and Nvidia Titan X (Pascal) 12GB GPU. The results are shown in Figure 6. In this setup *LaT-5* and *LaT-7* crashes with VGG16, and *AaT* crashes with ResNet50.