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

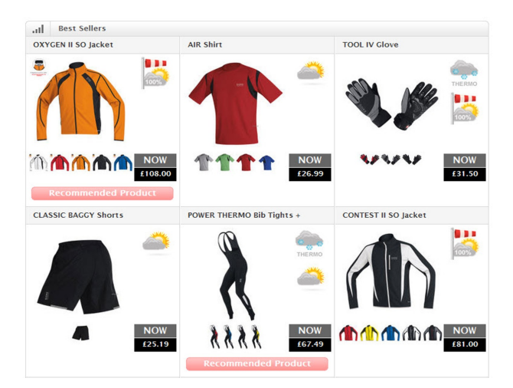
Supun Nakandala and Arun Kumar
University of California, San Diego
{snakanda, arunkk}@eng.ucsd.edu

1. Summary

Deep CNNs achieve near-human accuracy in image understanding tasks.

Observation: In many applications powered primarily by structured features, useful image data ignored or underutilized

Opportunity: Feature transfer from pre-trained deep CNNs is a powerful way to cheaply exploit image data



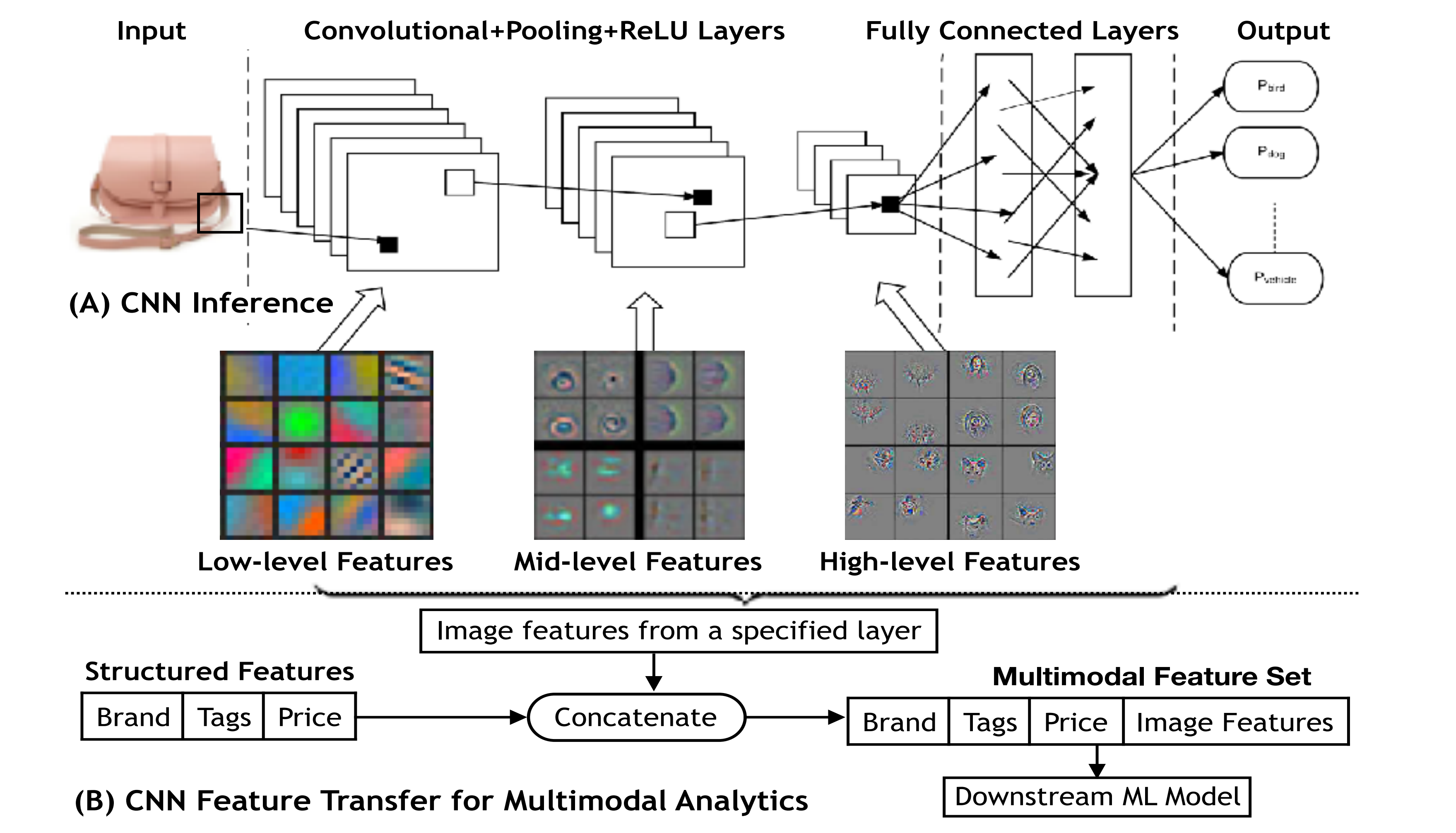
Problem: Effective feature transfer requires exploratory comparison of different layers; requires managing feature materialization at scale

Our Approach: Elevate the feature transfer workload to a declarative level; automate optimization of feature materialization trade-offs at scale to improve efficiency and reliability

Results: Our system, Vista, improves reliability and reduces runtimes by up to 90%

2. Background and Example

The key technical reason for deep CNNs' success is how they extract a hierarchy of relevant features from images.



Example Use Case: A data scientist working for an online fashion retailer wants to build a product recommendation model. She can now use product images which caries important information such as customer's stylistic preferences.

Has to explore multiple layers. Will write activations of each layer into files, join with structured features, and train the downstream ML model for each layer, e.g., using TensorFlow for CNN inference and Spark for downstream ML model training.

Iteratively performing CNN inference for each layer is inefficient. Repeated computations!

CNN features can be big (orders of magnitude larger than raw images). Can cause efficiency issues due to disk spills or even system crashes!

3. Problem Formalization

Inputs: Two tables $T_{str}(\underline{ID}, X)$ and $T_{img}(\underline{ID}, I)$, $X \in \mathbb{R}^{ds}$ is the structured feature vector, and I are raw images (say, as files on HDFS).

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)

Target: Train M for each of the $|L|$ feature vectors obtained by concatenating X with the respective feature layers obtained by partial CNN inference.

More Formally:

$$\forall\, l \in L:$$
$$T'_{img,l}\left(\underline{ID}, g_l\left(\hat{f}_l(I)\right)\right) \leftarrow \text{Apply } g_l \circ \hat{f}_l \text{ to } T_{img}$$
$$T'_l\left(\underline{ID}, X'_l\right) \leftarrow T_{str} \bowtie T'_{img,l}$$
$$\text{Train } M \text{ on } T'_l \text{ with } X'_l \equiv [X, \left(\hat{f}_l(I)\right)]$$

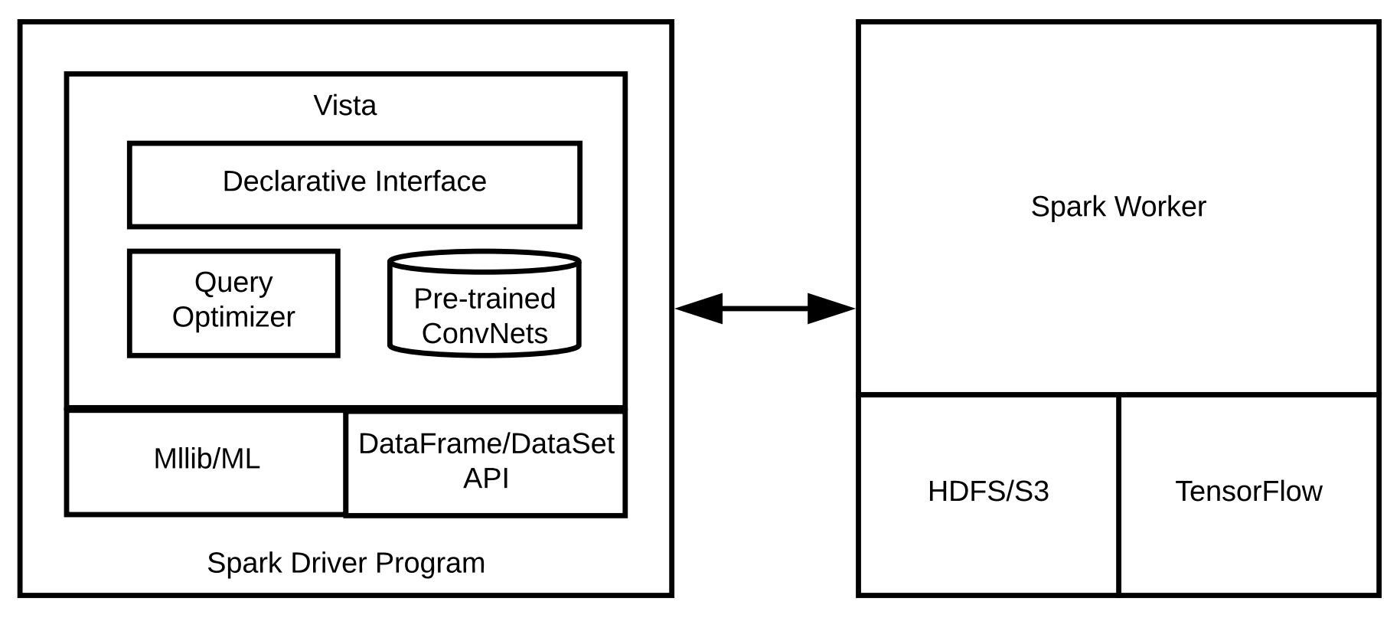
L	List of layers interested in exploring
$T_{str}(\underline{ID}, X)$	Structured data table
$T_{img}(\underline{ID}, I)$	Image data table
$\hat{f}_l$	CNN inference operator to extract features for layer l starting from images
g_l	CNN feature flattening operator
$\hat{f}_{l1 \to l2}$	Partial CNN inference operator for extracting CNN features for layer $l2$ starting from layer $l1$ CNN features.

4. System Overview

Vista is implemented as a library on top of the Spark-TensorFlow combine.

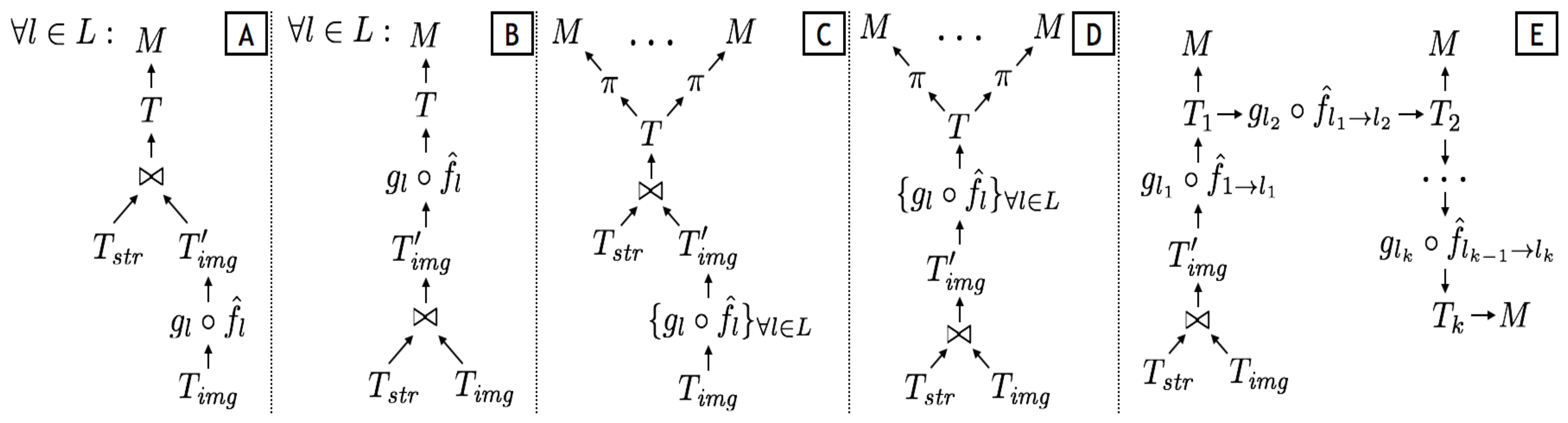
Declarative API to specify the feature transfer workload.

Optimizer picks an optimal plan.



5. System Optimizations

(1) Logical Plan Decisions



A	Current dominant approach of performing CNN inference.
B	Performing the CNN inference after the join can reduce the join overhead.
C	Bulk Inference
D	Bulk with CNN inference is performed after the join.
E	Staged Inference

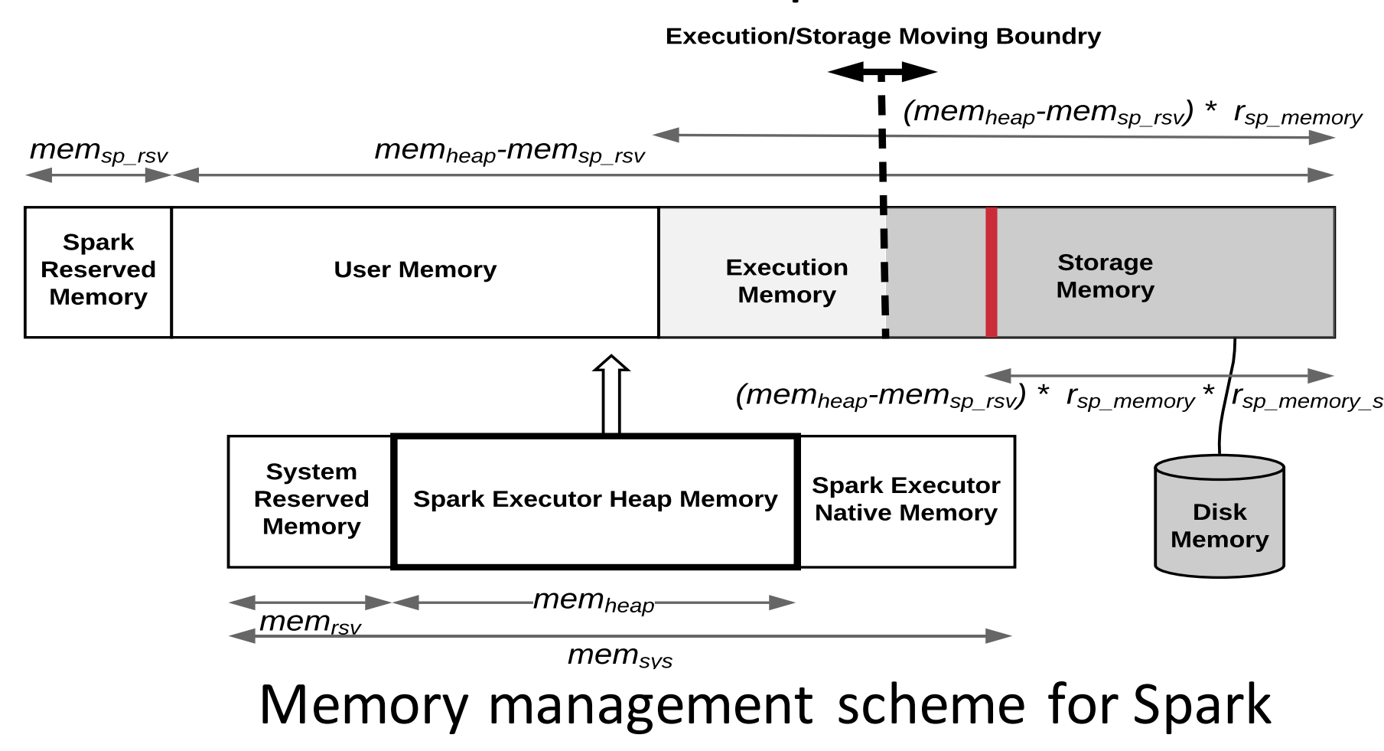
In brief: The new Bulk and Staged inference plans avoid redundant partial CNN inference; Staged with join pushed below also has lower memory pressure

(2) System Configuration Choices

Mismanaged memory can cause system crashes and excessive disk spills that raise runtimes in Spark.

Has to pick four configuration values:

1. CPUs per executor
2. JVM heap size
3. User memory size
4. Number of data partitions


Execution/Storage Moving Boundry
mem_sp_rsv
mem_heap-mem_sp_rsv
(mem_heap-mem_sp_rsv) * r_sp_memory
Spark Reserved Memory
User Memory
Execution Memory
Storage Memory
(mem_heap-mem_sp_rsv) * r_sp_memory * r_sp_memory_storage
System Reserved Memory
Spark Executor Heap Memory
Spark Executor Native Memory
Disk Memory
mem_rsv
mem_heap
mem_sys

Memory management scheme for Spark

In brief: Vista uses information about CNN operator characteristics such as input/output dimensions, runtime memory footprints and estimates of intermediate data sizes to pick system configuration values.

(3) Physical Plan Decisions

Pick broadcast join or shuffle join? Depends on the size of T_{str}

Store intermediate data in serialized form? Depends on chance of disk spills

Vista Optimizer

We define simple optimization problem for minimizing runtime subject to a suite of memory constraints to ensure Spark will not crash

Line search algorithm to set all free parameters and execution plan decisions

6. Snapshot of Results

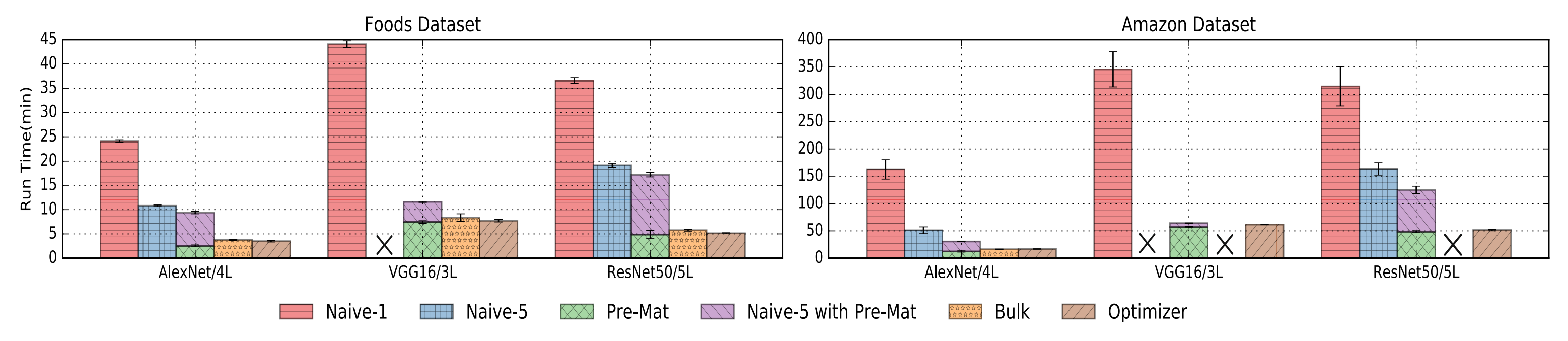
Naïve 1, Naïve 5: The current dominant approach. Corresponds to logical plan A.

Pre-Mat, Bulk: Advanced baselines.

Optimizer: Plan picked by Vista optimizer.

Datasets	Foods	Amazon
Description	Food nutrition facts (e.g. sugar, proteins)	Product information (e.g. price, brand)
# structured features	130	200
Image	Of the food item	Of the product
Target	Is food plant-based?	Is product popular?

Setup: Using 8 node Spark (2.2.0) cluster (8 x 2.00 GHz CPUs, 32 GB RAM, and 300 GB HDD per node) and TensorFlow (1.3.0). Downstream ML: logistic regression for 10 iterations.



Takeaway: Vista improves reliability of deep CNN feature transfer workload and reduces runtimes up to 90% at scale

Project web page with tech report, code, and data:

http://adalabucsd.github.io/vista.html