Introduction building a distributed neural network on Apache Spark with Analytics Zoo

ODSC East 2019 Workshop

DELL EMC

Quiz

- Familiarity with Apache Spark
- Familiarity with Deep Learning
- Familiarity with TensorFlow and Keras

Links

- DockerHub: https://hub.docker.com/r/dellai/odsc-east-2019
- GitHub: https://github.com/Dell-AI/ODSC-east-2019

#ODSC

BOSTON
APR 30 - MAY 3

Introduction to building a distributed neural network on Apache Spark with BigDL and Analytics Zoo

Bala Chandrasekaran

Technical Staff,
Dell Technologies

#ODSC

BOSTON

APR 30 - MAY 3

Introduction to building a distributed neural network on Apache Spark with BigDL and Analytics Zoo

Andrew Kipp

Data Scientist,
Dell Technologies

Yuhao Yang

Staff Software Engineer, Intel
Contributor to Analytics Zoo

Docker instructions

- Docker pull
  - `docker pull dellai/odsc-east-2019:1.0`
- Docker run
  - ```
    sudo docker run -it --rm -p 12345:12345 -p 12346:12346
      -e NotebookPort=12345
      -e NotebookToken="your-token"
      dellai/odsc-east-2019:1.0 bash
    ```
- Extract data
  - `cd ODSC-east-2019/datasets`
  - `./extract.sh`
  - `cd -`
- Run notebook
  - `./start-notebook.sh`

DELLEMC
    ```

Cautions ⚠ and notes

- Docker storage is not persistent. Do not take notes in the notebook
  - Unless you can create your own *fork* and *push* before closing the container
- Spark and deep learning are memory intensive. We have tested this 4GB laptop. A few tips:
  - Always shutdown one kernel before running another
  - If you get connection error to Java server, it is most likely due to lack of sufficient memory. Restart the notebook and try with a smaller dataset or less epochs
- If you are running on a Windows system
  - You must disable firewall to run notebooks (or configure it to allow the specific ports)
  - Configure Docker memory to be at least 4GB or more

Agenda

For the next 3 to 4 hours !

- Concepts
  - Apache Spark
  - Deep Learning
  - Analytics Zoo
- Notebooks (with light coding)
  - MNIST
  - Image Augmentation
  - Transfer learning
  - Recommendation using NCF
  - Object detection
  - Anomaly detection (if time permits)
  - Inference: Detecting diseases in Chest X-rays (if time permits)
  - Image Similarity (if time really permits)

About

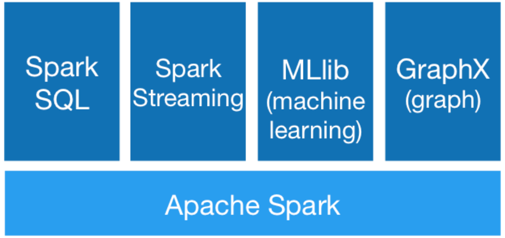
- Purpose
  - Introduction to deep learning in Apache Spark using Analytics Zoo
  - Focus on Image processing, but also cover NCF and Anomaly Detection
- Pre-Requisites
  - Python
  - Jupiter Notebook

Apache Spark*

Apache Spark

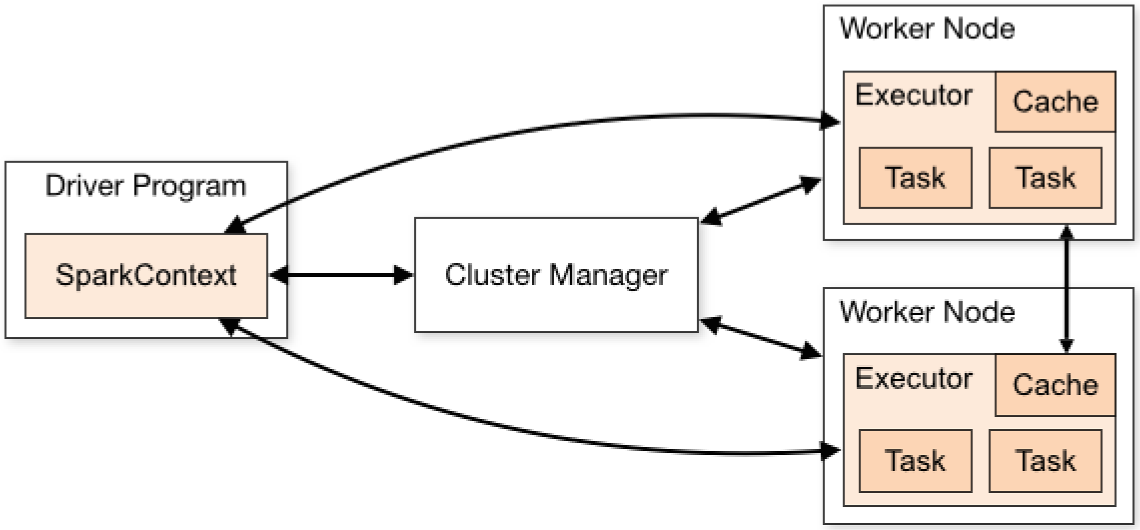
Unified analytics engine for large-scale data processing

- Unified
  - SQL
  - Streaming
  - Machine Learning
- Speed
  - High performance for both batch and streaming data
  - In memory computation
- Flexible
  - Supports Java, Scala, Python, R, and SQL
  - Runs anywhere: Stand-alone, Hadoop, YARN, Mesos, Kubernetes
  - Supports various persistent storage



https://spark.apache.org/

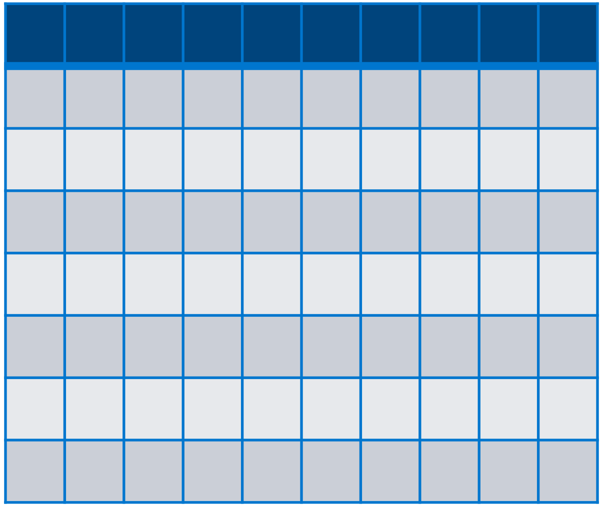
Spark Architecture



Source: https://spark.apache.org/docs/latest/cluster-overview.html

Spark DataFrames

- Most common way to represent data in Spark
- Schema defines the columns and types of columns
- Supports reading from JSON, Parquet files, Hive, etc
  - With Analytics Zoo and other new APIs (Spark 2.4) you can read images as Spark DataFrames
- Partitions
  - DataFrames are broken into partitions and distributed across the cluster
  - Each worker can work on a partition
  - Abstracted from the end user/developer



- origin: `StringType` (represents the file path of the imag
- height: `IntegerType` (height of the image)
- width: `IntegerType` (width of the image)
- nChannels: `IntegerType` (number of image channels)
- mode: `IntegerType` (OpenCV-compatible type)
- data: `BinaryType` (Image bytes in OpenCV-compatible

Source: https://spark.apache.org/docs/latest/ml-datasource#image-data-source

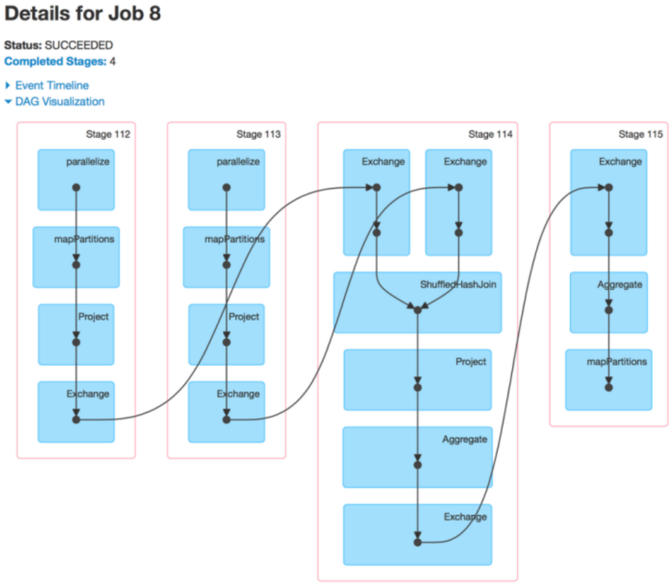
Resilient Distributed Datasets (RDDs)

- Low level API
  - No longer commonly used after DataFrames are introduced in Spark 2.x
- Fine grained control over physical distribution and transformations
- DataFrames use RDDs in the background

DELL EMC

Spark Transformations

- Data structures in Spark are *immutable*
- *Transformation* are used to 'change' data
- Basic Operations
  - df.printSchema()
  - column, count, expr,
- Example transformations:
  - Aggregations: count, min, max, first, last, sum, avg
  - Joins, filters and maps
- Lazy evaluation
  - Wait until the last minute to execute
  - Build a plan
- Directed Acrylic Graphs



Source: https://databricks.com/blog/2015/06/22/understanding-your-spark-application-through-visualization.html

DELLEMC

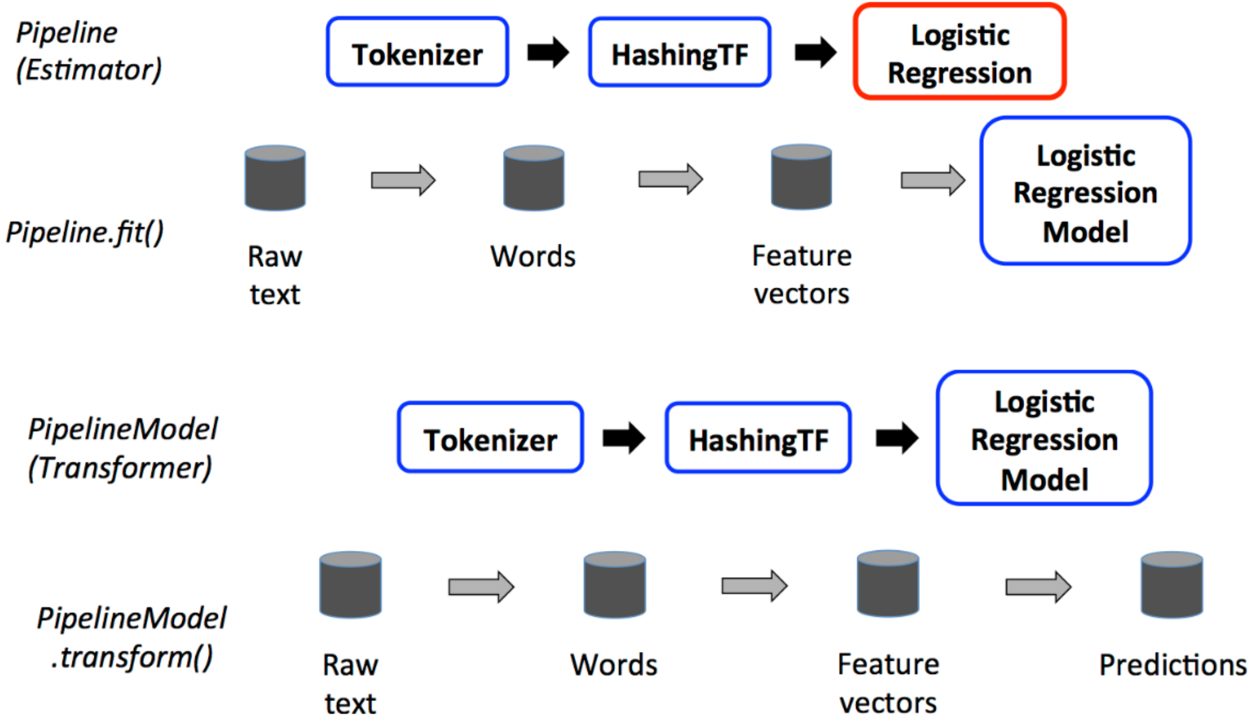
Spark ML Pipelines - Concepts

- Transformer: An algorithm to transform one DataFrame into another DataFrame
- Estimator: An algorithm which can be *fit* on a DataFrame to produce a Transformer.
- Pipeline: A Pipeline chains multiple Transformers and Estimators together to specify an ML workflow
- Parameter: All Transformers and Estimators share a common API for specifying parameters

Source: https://spark.apache.org/docs/latest/ml-pipeline.html

DELL EMC

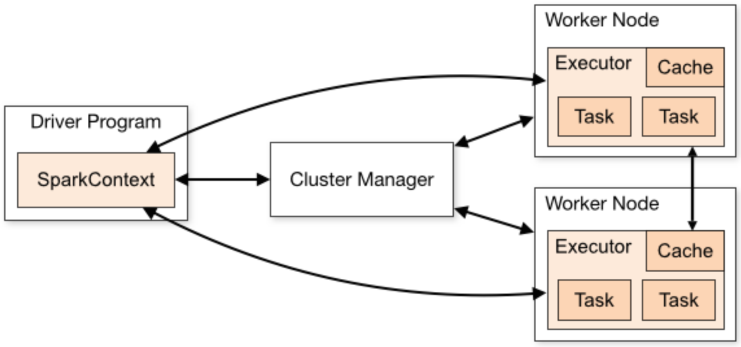
Spark ML Pipelines - Illustration



Source: https://spark.apache.org/docs/latest/ml-pipeline.html

Spark Submit – Key Parameters

Parameter	Description
--master local --master local[k] --master local[*]	Run Spark locally with one worker thread (i.e. no parallelism at all), k threads, as many threads as logical cores
spark.driver.cores	The number of cores for the driver
spark.driver.memory	Amount of memory for the driver
spark.executor.memory	Amount of memory to use per executor process
spark.executor.cores	The number of cores in each executor

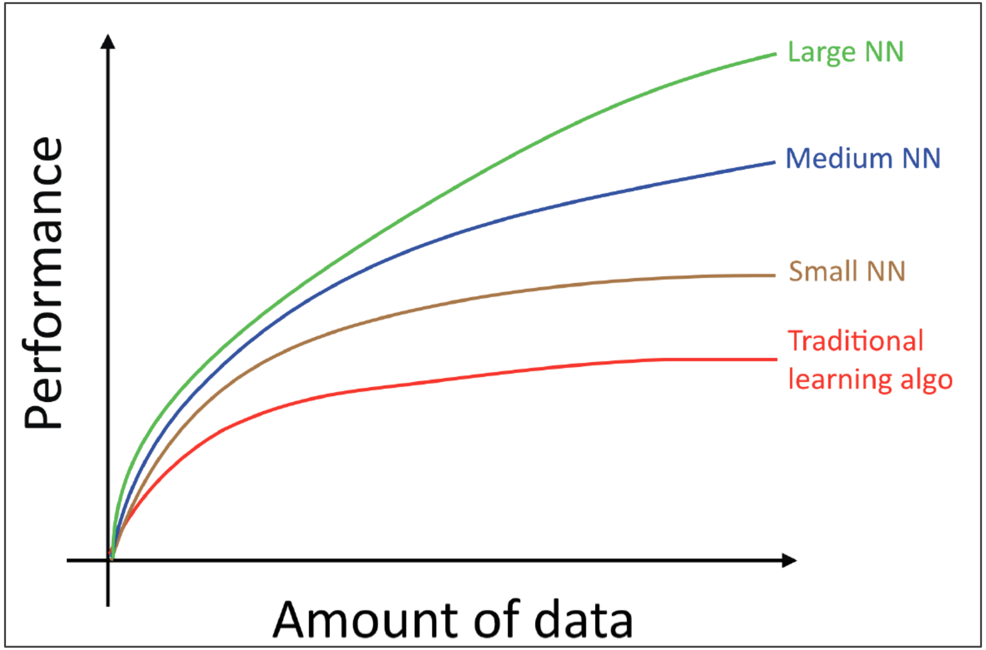


Source: https://spark.apache.org/docs/latest/submitting-applications.html

Deep Learning and Artificial Neural Network

DELL EMC

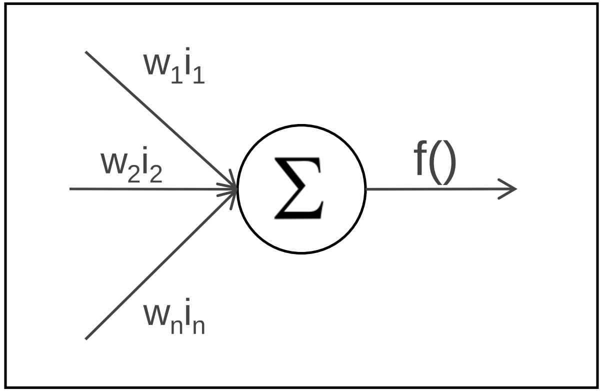
Motivation

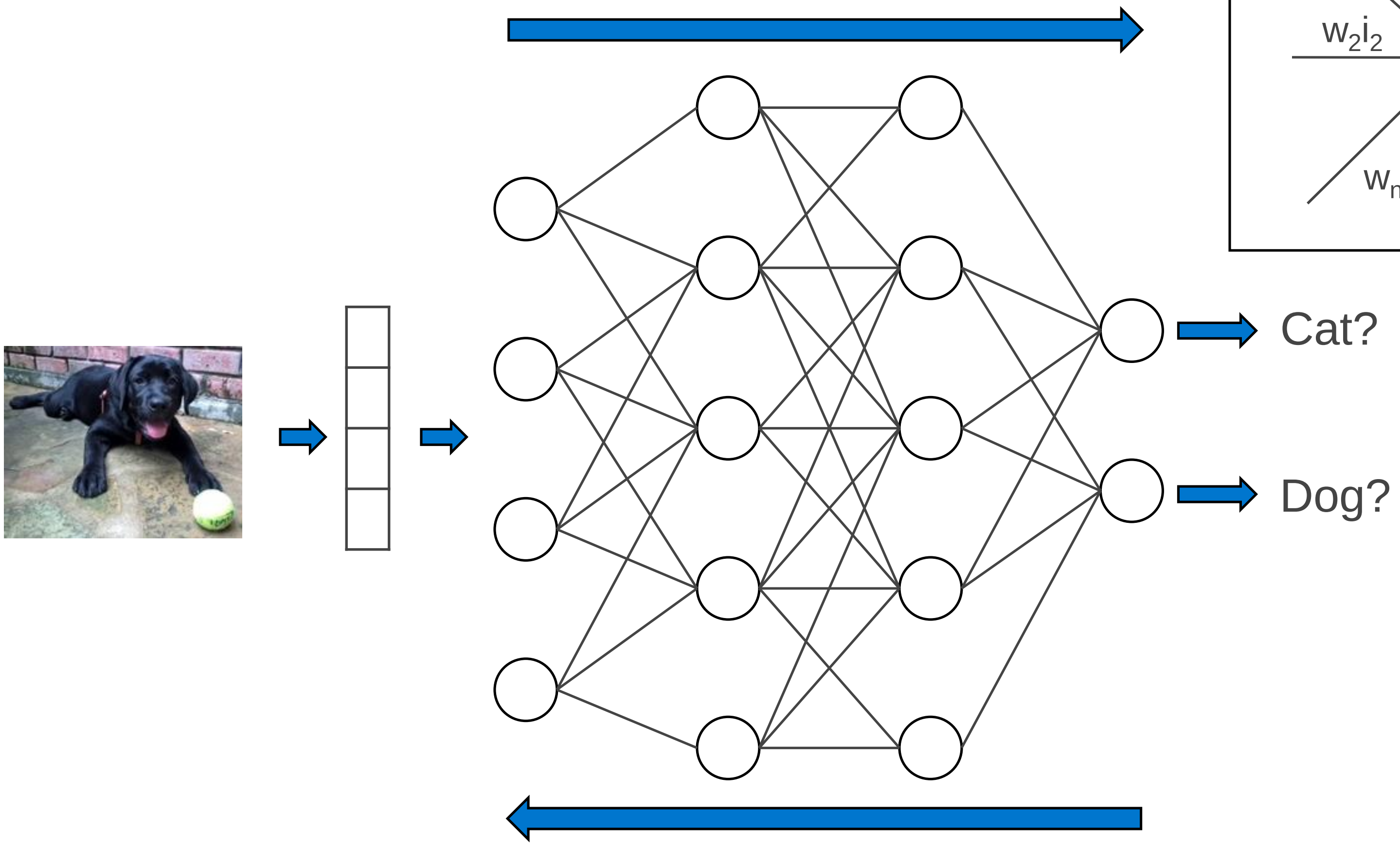


"Machine Learning Yearning", Andrew Ng, 2016

DELLEMC

Multi-layer Perceptron

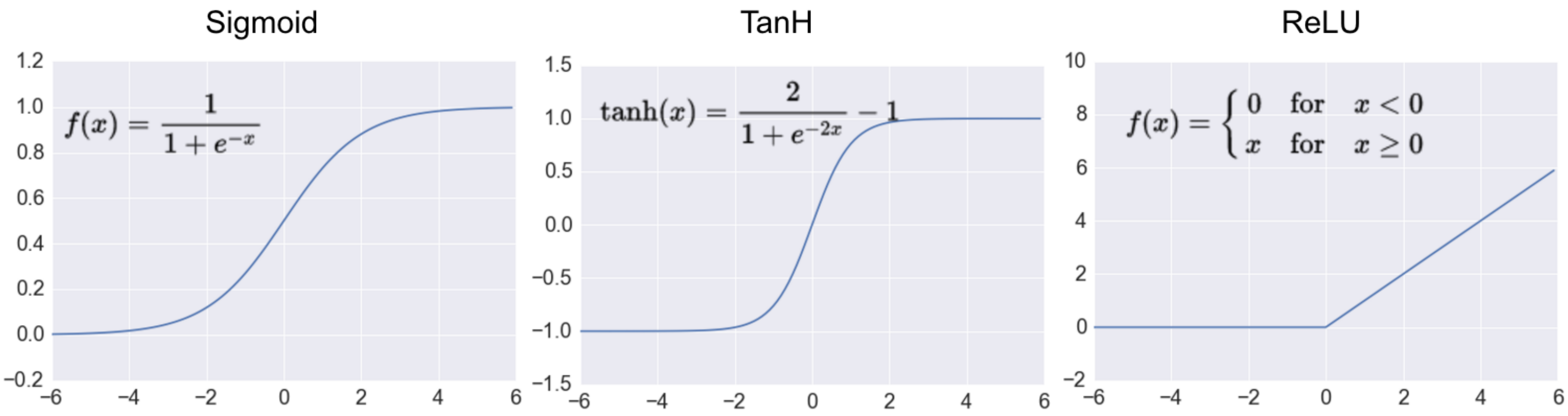




DELLEMC

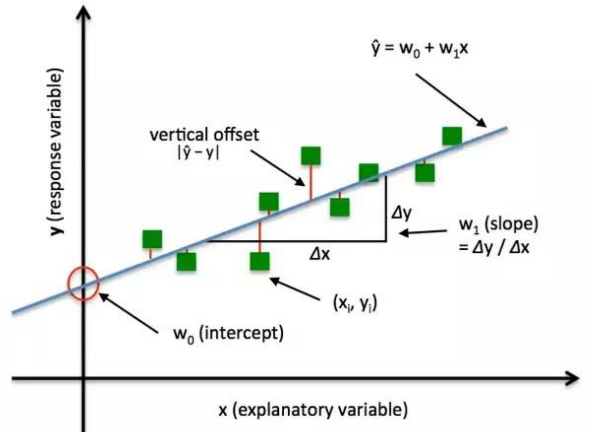
Activation Function

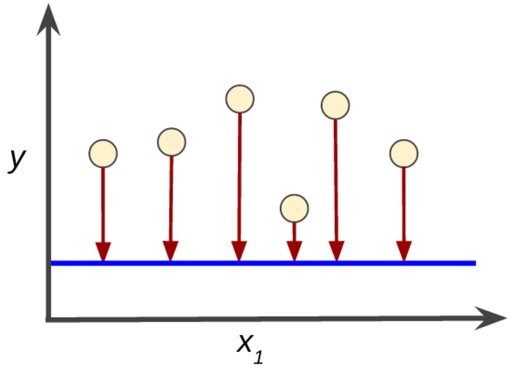
- Activation function “activates” a neuron

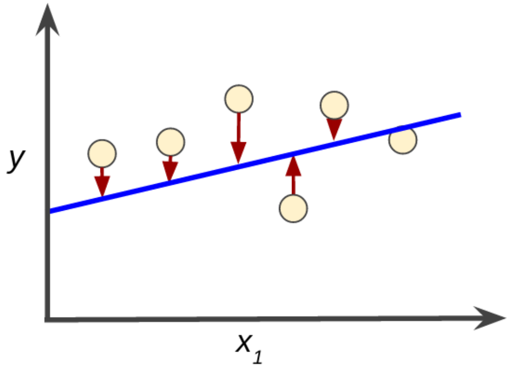


Source: [Practical Introduction to Deep Learning]

Loss Function



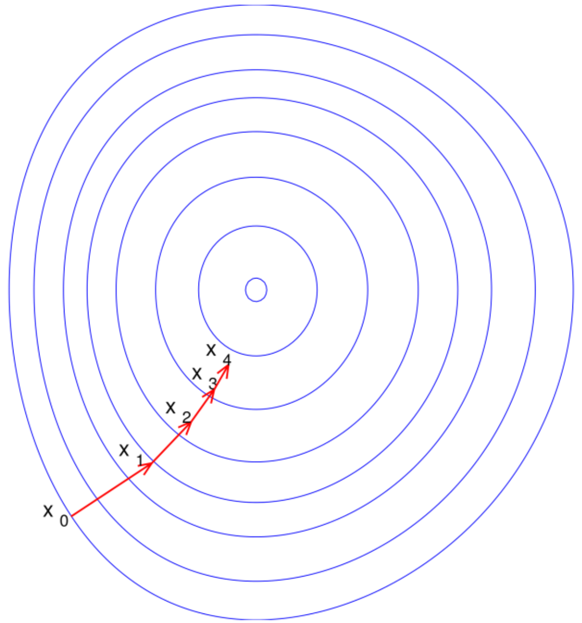




Sources: *[Algorithmia: Introduction to Loss Functions](), [Google Machine Learning Crash Course]()*

Optimizer

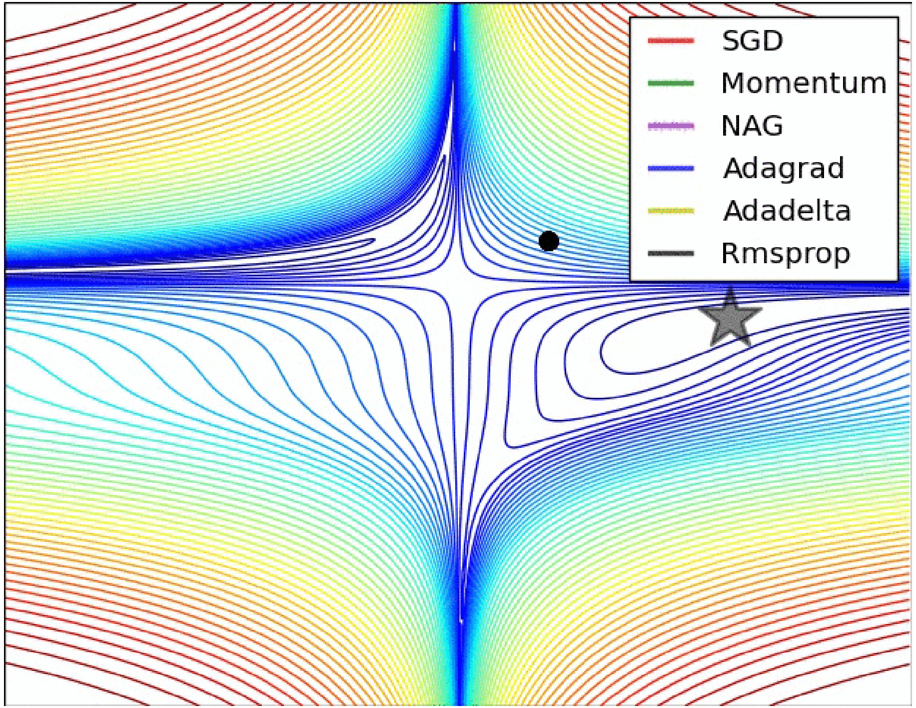
- Optimizers determine how your model trains
  - Ties the loss function and model parameters
- Popular optimizer: Gradient decent
- Mini batch gradient decent
  - Updates the model for every mini-batch of training samples

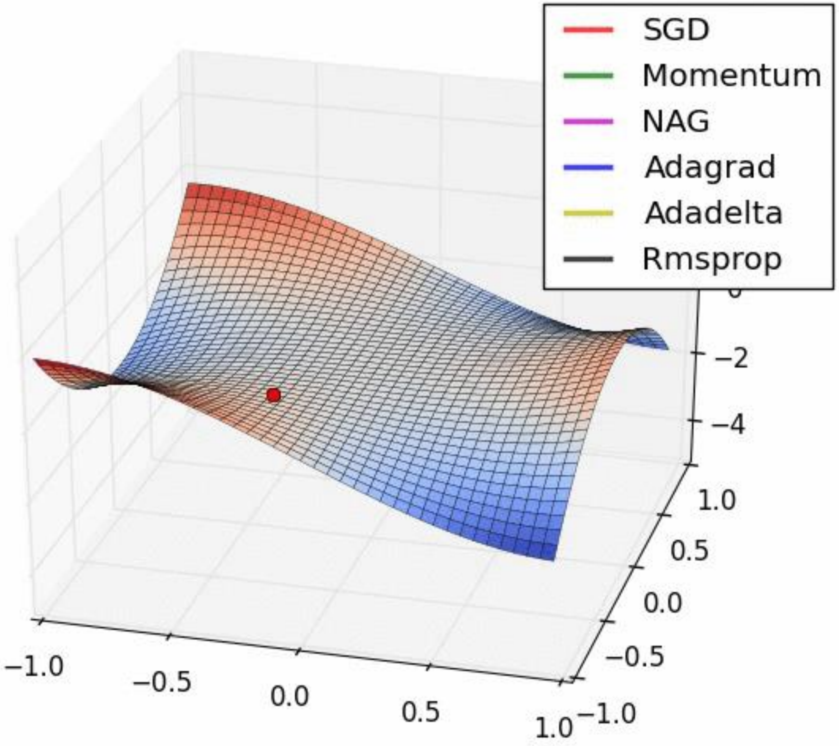


Gradient decent

Source: https://en.wikipedia.org/wiki/Gradient_descent

Optimizer

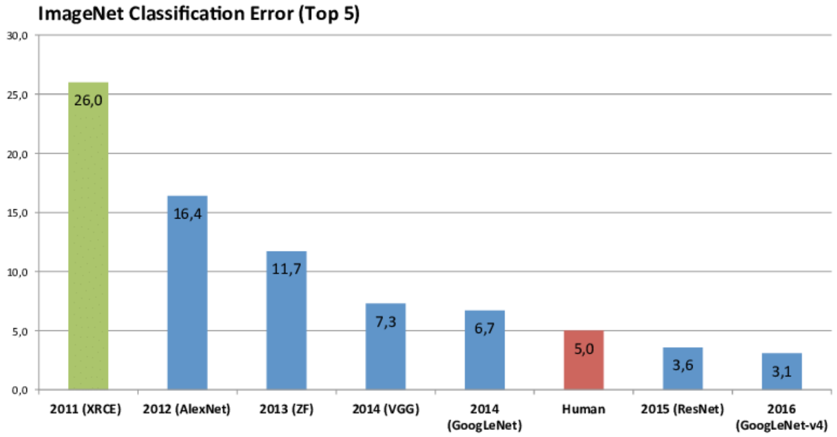




Source: Sebastian Ruder (2016). *An overview of gradient descent optimisation algorithms. arXiv preprint arXiv:1609.04747.*

Popular convolutional neural networks for image classification

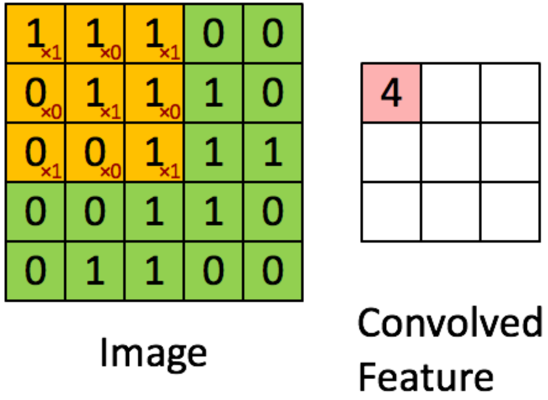
- VGG
- Inception (GoogLeNet)
- ResNet
- DenseNet

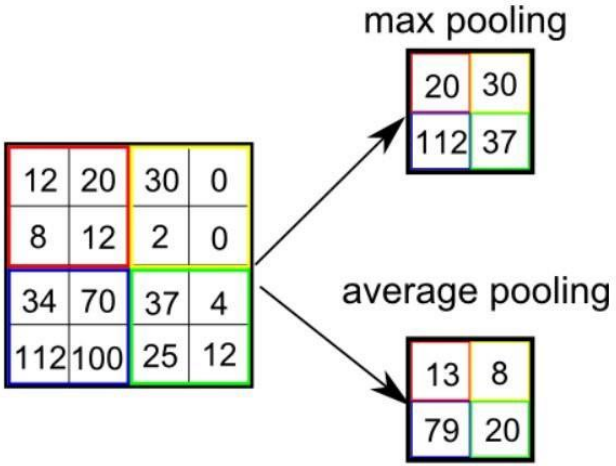


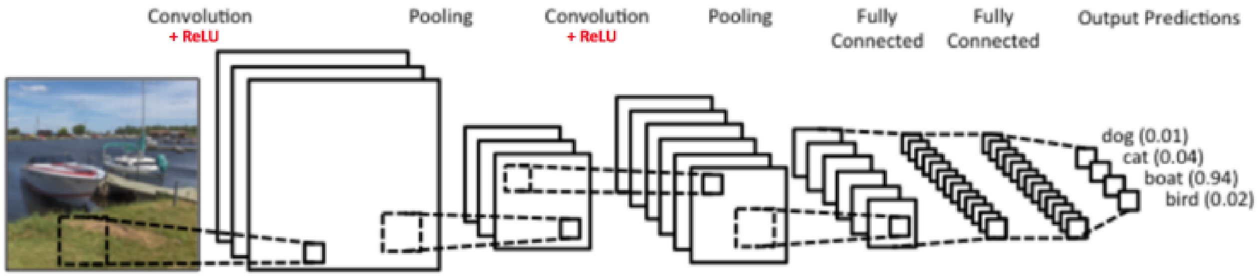
Source: https://www.researchgate.net/figure/Winner-results-of-the-ImageNet-large-scale-visual-recognition-challenge-LSVRC-of-the_fig7_324476862

DELL EMC

Convolutions

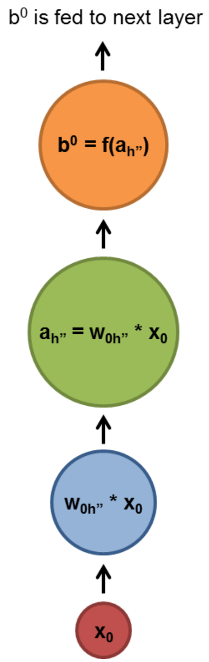






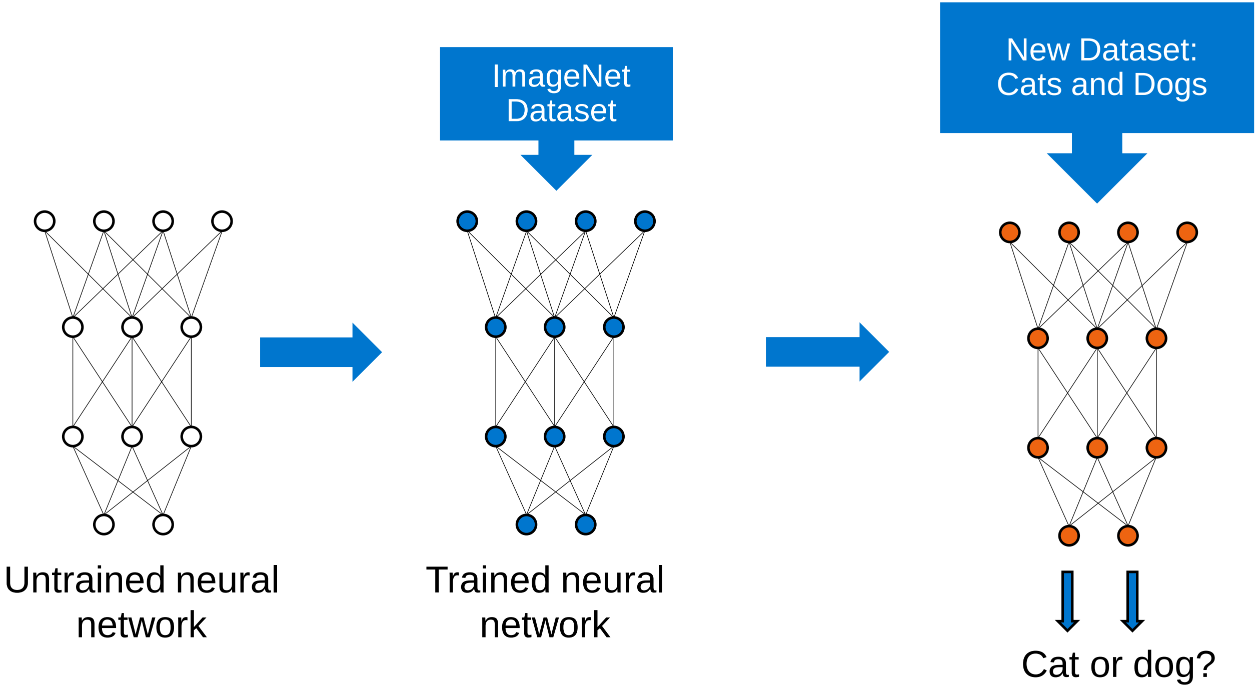
Sources: [Comprehensive Guide to CNNs](), [Intuitive Explanation of CNNs]()

Recurrent Neural Network



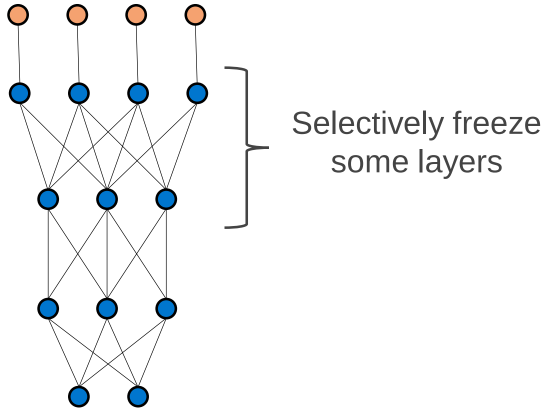
Source: [Skymind A.I. Wiki: RNNs and LSTMs]

Transfer Learning



Transfer Learning – Typical Operations

- Add a new input layer
- Remove the old classification layer
- Add a new classification layer
- Freeze some layers



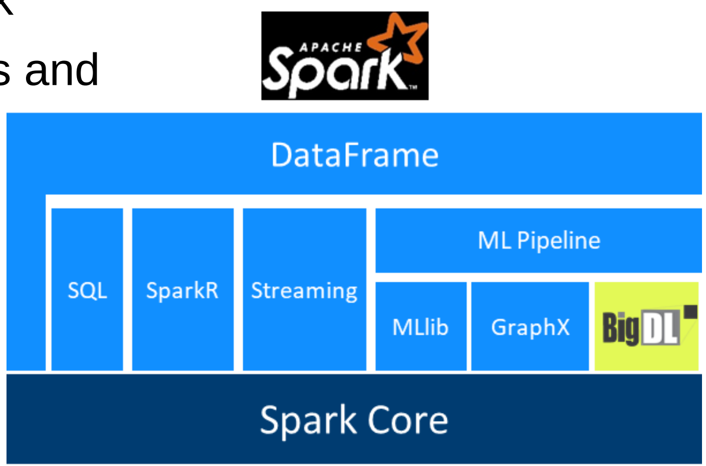
Analytics Zoo

DELL EMC

BigDL

Bringing Deep Learning To Big Data Platform

- Distributed deep learning framework for Apache Spark*
- Make deep learning more accessible to big data users and data scientists
  - Write deep learning applications as standard Spark programs
  - Run on existing Spark/Hadoop clusters (no changes needed)
- Feature parity with popular deep learning frameworks
  - E.g., Caffe, Torch, Tensorflow, etc.
- High performance (on CPU)
  - Built-in Intel MKL and multi-threaded programming
- Efficient scale-out
  - Leveraging Spark for distributed training & inference



https://github.com/intel-analytics/BigDL

https://bigdl-project.github.io/

Analytics Zoo

Unified Analytics + AI Platform for Big Data

Distributed TensorFlow, Keras and BigDL on Apache Spark

Reference Use Cases	Anomaly detection, sentiment analysis, fraud detection, chatbot, sequence prediction, etc.
Built-In Deep Learning Models	Image classification, object detection, text classification, recommendations, sequence-to-sequence, anomaly detection, etc.
Feature Engineering	Feature transformations for • Image, text, 3D imaging, time series, speech, etc.
High-Level Pipeline APIs	• Distributed TensorFlow and Keras on Spark • Native deep learning support in Spark DataFrames and ML Pipelines • Model serving API for model serving/inference pipelines
Backends	Spark, TensorFlow, Keras, BigDL, OpenVINO, MKL-DNN, etc.

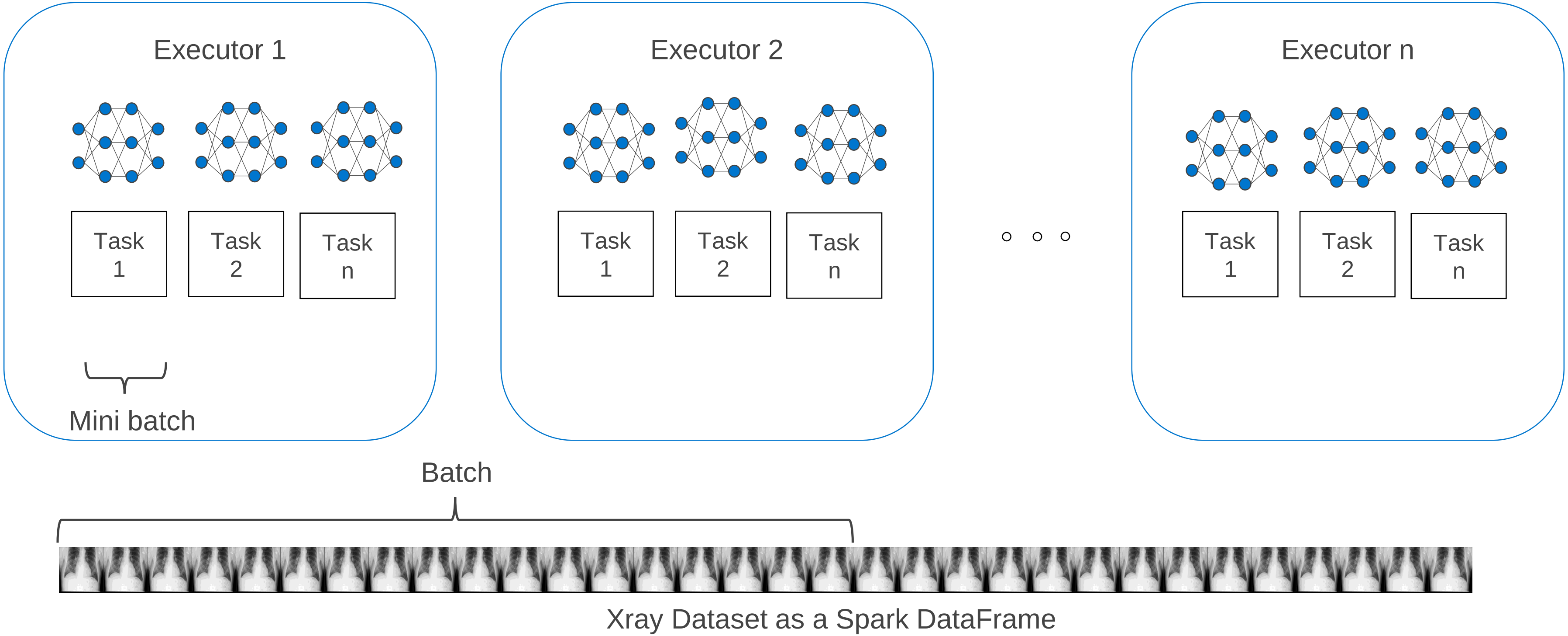
https://github.com/intel-analytics/analytics-zoo/

https://analytics-zoo.github.io/

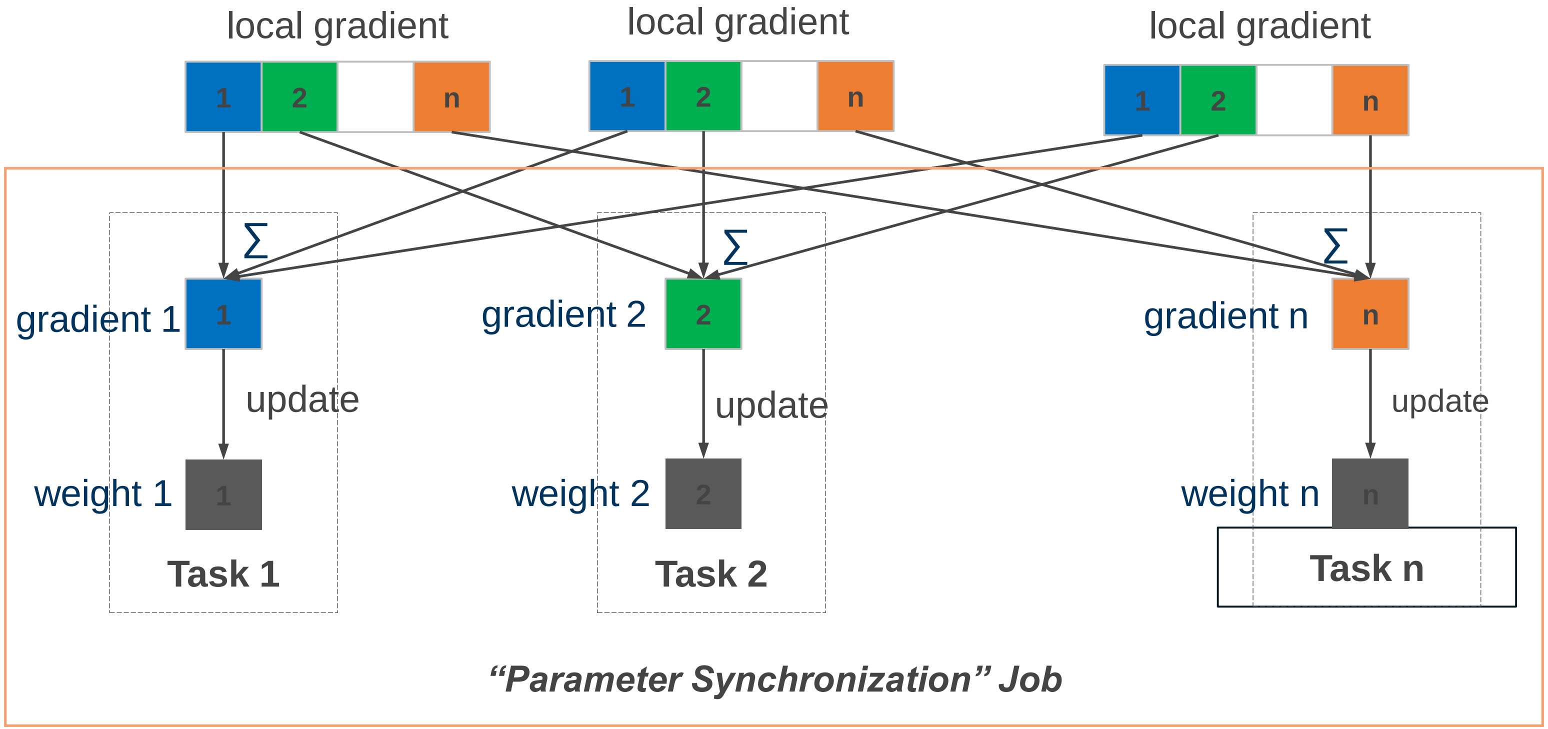
Analytics Zoo

- Build end-to-end deep learning applications for big data
  - Distributed TensorFlow on Spark
  - Keras-style APIs (with autograd & transfer learning support)
  - nnframes: native DL support for Spark DataFrames and ML Pipelines
  - Built-in feature engineering operations for data preprocessing
- Productionize deep learning applications for big data at scale
  - Model serving APIs (w/ OpenVINO support)
  - Support Web Services, Spark, Storm, Flink, Kafka, etc.
- Out-of-the-box solutions
  - Built-in deep learning models and reference use cases

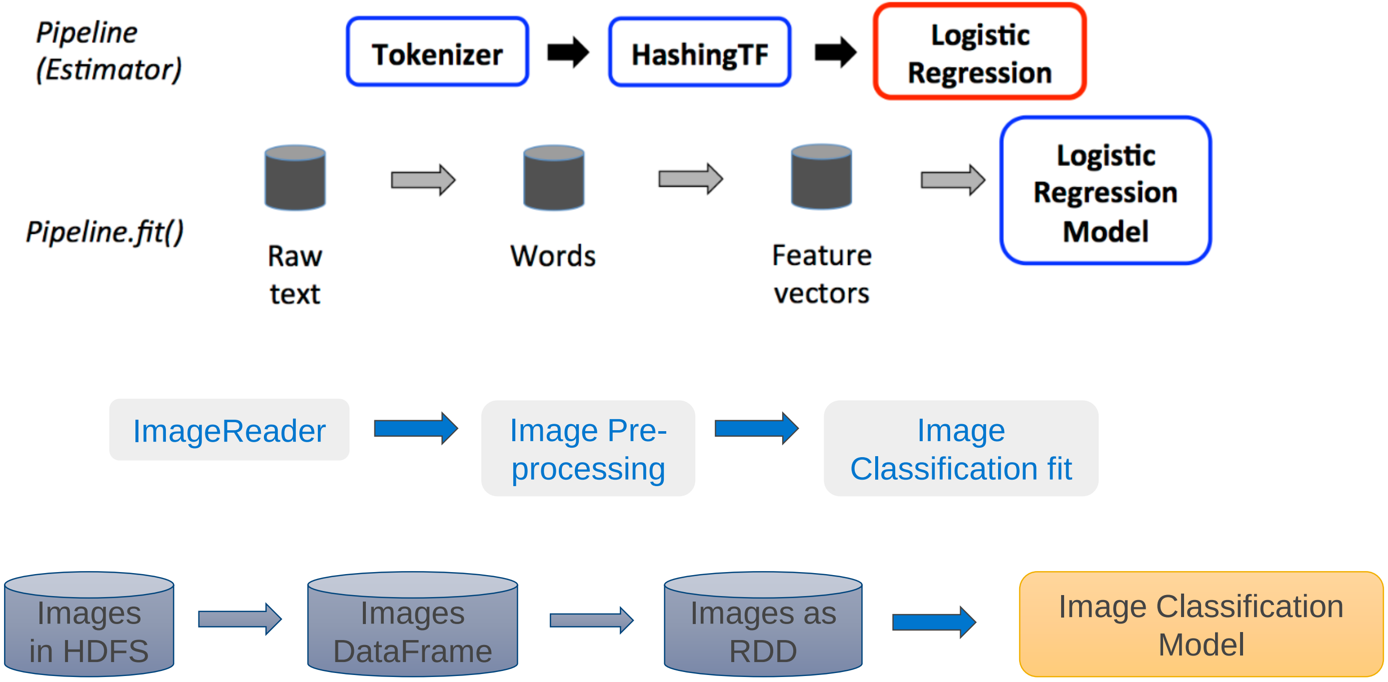
Distributed training in BigDL



Parameter Synchronization



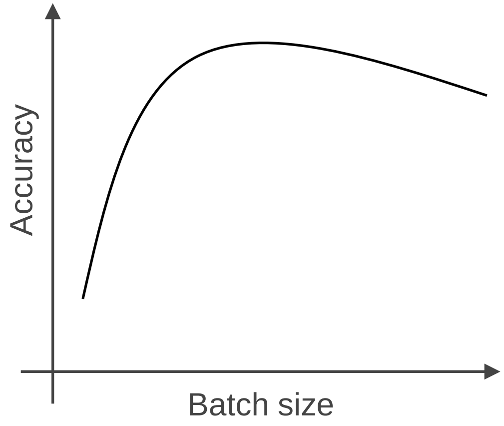
Deep Learning Pipeline in Analytics Zoo



Notes

A note on Batch Size and impact on Scalability

- Two factors:
  - Batches per thread (mini-batch): Number of images in a iteration per worker
  - Global batch size: Batches per thread * Number of workers
- Batch size is a critical parameter for model accuracy as well as for distributed performance
  - Increasing the batch size increases parallelism
  - Increasing the batch size may lead to a loss in generalization performance especially



DELL EMC

Let’s code !

About the Container

Version 1.1

- `/opt/work/ODSC-east-2019`
  - `./docker` – Contains files to build the docker
  - `./start-notebook.sh` – Script to start the notebook
  - `./datasets` – Contains the datasets required for this workshop. Make sure you run `./extrach.sh`
  - `./models` – Contains required models for this workshop
  - `./Final-notebooks` – Out of the box working examples
    - `./Example1-MNIST` – Bug – Make sure the paths are correct
    - `./Example5-ObjectDetection` – Bug – Make sure the paths are correct
    - `./Example6-ImageSimilarity` – Does not contain the dataset (23 GB)
    - `./Example8-Xray` – Training does not contain the NIH chest xray dataset (~120,000 xray images)
  - `./Workshop-notebooks` – We will use these ‘incomplete’ notebooks for this workshop

MNIST

How to add Tensorboard

- `docker ps`
  - Get your container name. Something like *clever_liskov*
- `docker exec -it clever_liskov bash`
- `tensorboard --logdir=/tmp/mnist_log/mnist/train --port 12346`

Neural Collaborative Filtering

What is recommendation?

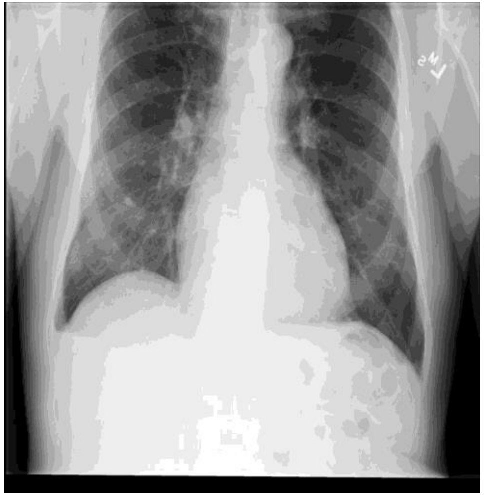
Source:

Chest X-rays

DELL EMC

Predicting diseases in Chest X-rays

- Develop a ML pipeline in Apache Spark and train a deep learning model to predict disease in Chest X-rays
  - An integrated ML pipeline with Analytics Zoo on Apache Spark
  - Demonstrate feature engineering and transfer learning APIs in Analytics Zoo
  - Use Spark worker nodes to train at scale



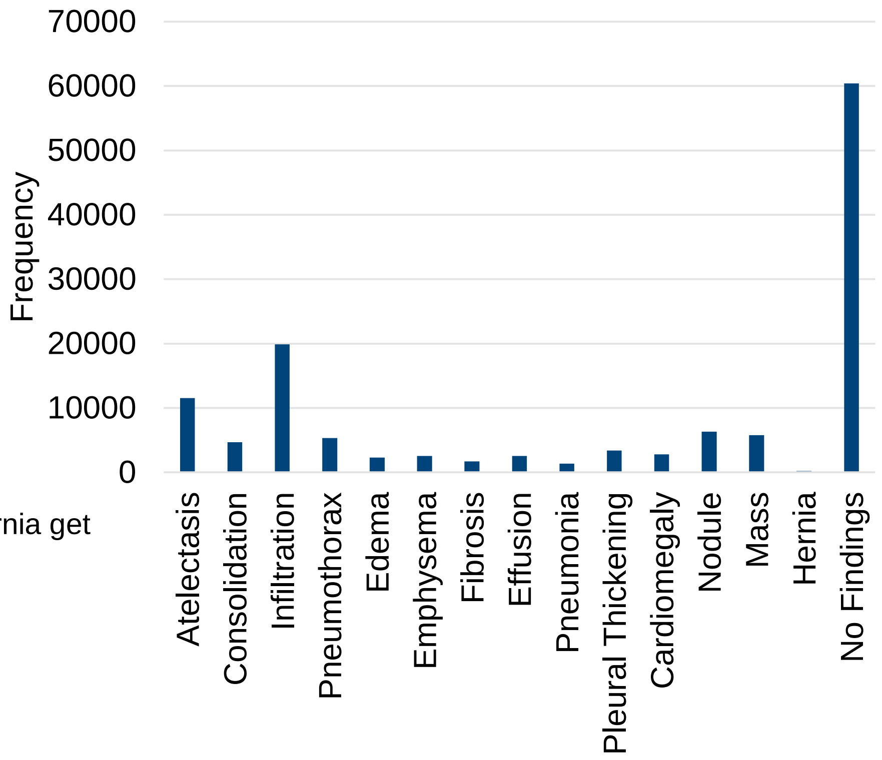
- CheXNet
  - Developed at Stanford University, CheXNet is a model for identifying thoracic pathologies from the NIH ChestXray14 dataset
  - https://stanfordmlgroup.github.io/projects/chexnet/

X-ray dataset from NIH

- 112,120 images from over 30000 patients
- Multi label (14 diseases)

  00000013_005.png,Emphysema | Infiltration | Pleural_Thickening
  00000013_006.png,Effusion|Infiltration
  00000013_007.png,Infiltration
  00000013_008.png,No Finding

- Unbalanced datasets
  - Close to 50% of the images have ‘No findings’
  - Infiltration get the most positive samples (19894) and Hernia get the least positive samples (227)



DELLEMC

Let’s build the model

Read the X-ray images

↓

Feature Engineering

↓

Define the model

↓

Define optimizer

↓

Train the model

Read the X-ray images as Spark DataFrames

- Read the X-ray images
- Feature Engineering
- Define the model
- Define optimizer
- Training the model

- Initialize NNContext and load X-ray images into DataFrames using NNImageReader

```
from zoo.pipeline.nnframes import NNImageReader
imageDF = NNImageReader.readImages(image_path, sc,
                                   resizeH=256, resizeW=256,
image_codec=1)
```

- Process loaded X-ray images and add labels (another DataFrame) using Spark transformations

```
trainingDF = imageDF.join(labelDF, on="Image_Index",
             how="inner")
```

Feature Engineering – Image Pre-processing

- Read the X-ray images
- ↓
- Feature Engineering
- ↓
- Define the model
- ↓
- Define optimizer
- ↓
- Training the model

In-built APIs for feature engineering using *ChainedPreprocessing* API

ImageCenterCrop(224, 224)

+

Random flip and brightness

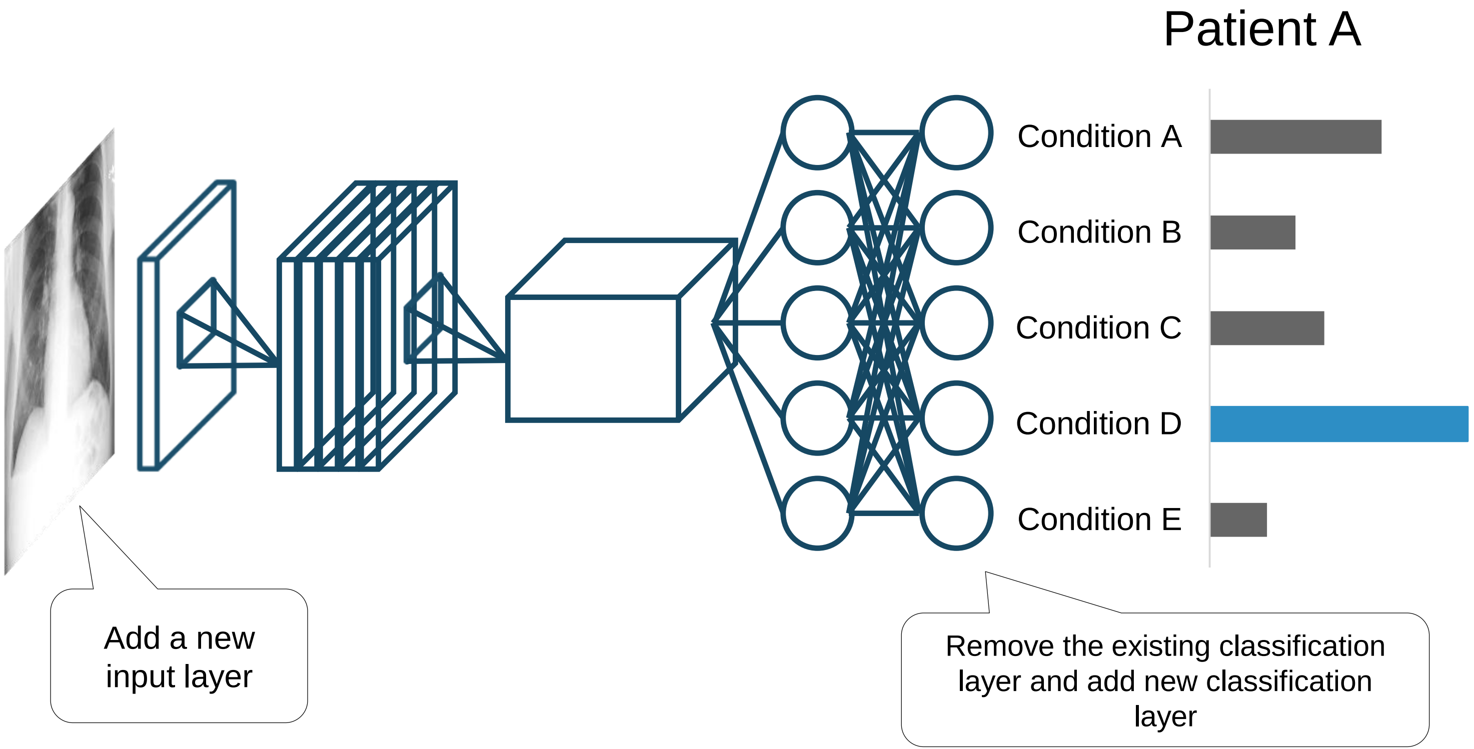
+

ImageChannelNormalize()

+

To Tensor()

Transfer Learning using ResNet-50 trained with ImageNet



DELLEMC

Defining the model with Transfer Learning APIs

- Read the X-ray images
- Feature Engineering
- Define the model
- Define optimizer
- Training the model

- Load a pre-trained model using *Net.load_bigdl*. The model is trained with ImageNet dataset
  - Inception
  - ResNet 50
  - DenseNet
- Remove the final *softmax* layer of ResNet-50
- Add new input (for resized x-ray images) and output layer (to predict the 14 diseases). Activation function is *Sigmoid*
- Avoid overfitting
  - Regularization
  - Dropout

Defining the model with Transfer Learning APIs

- Read the X-ray images
- Feature Engineering
- Define the model
- Define optimizer
- Training the model

```
def get_resnet_model(model_path, label_length):

        full_model = Net.load_bigdl(model_path)

        model = full_model.new_graph(["pool5"])

        inputNode = Input(name="input", shape=(3, 224, 224))

        resnet = model.to_keras()(inputNode)

        flatten = GlobalAveragePooling2D(dim_ordering='th')(resnet)

        dropout = Dropout(0.2)(flatten)

        logits = Dense(label_length, W_regularizer=L2Regularizer
                (1e-1), b_regularizer=L2Regularizer(1e-1),
                activation="sigmoid")(dropout)

        lrModel = Model(inputNode, logits)

        return lrModel
```

Define the Optimizer

- Read the X-ray images
- Feature Engineering
- Define the model
- Define optimizer
- Training the model

- Evaluated two optimizers: SGD and Adam Optimizer
- Learning rate scheduler is implemented in two phases:
  - Warmup + Plateau schedule
  - Warmup: Gradually increase the learning rate for 5 epochs
  - Plateau: Plateau(**"Loss"**, factor=0.1, patience=1, mode=**"min"**, epsilon=0.01, cooldown=0, min_lr=1e-15)

DELL EMC

Train the model using ML Pipelines

Read the X-ray images

Feature Engineering

Define the model

Define optimizer

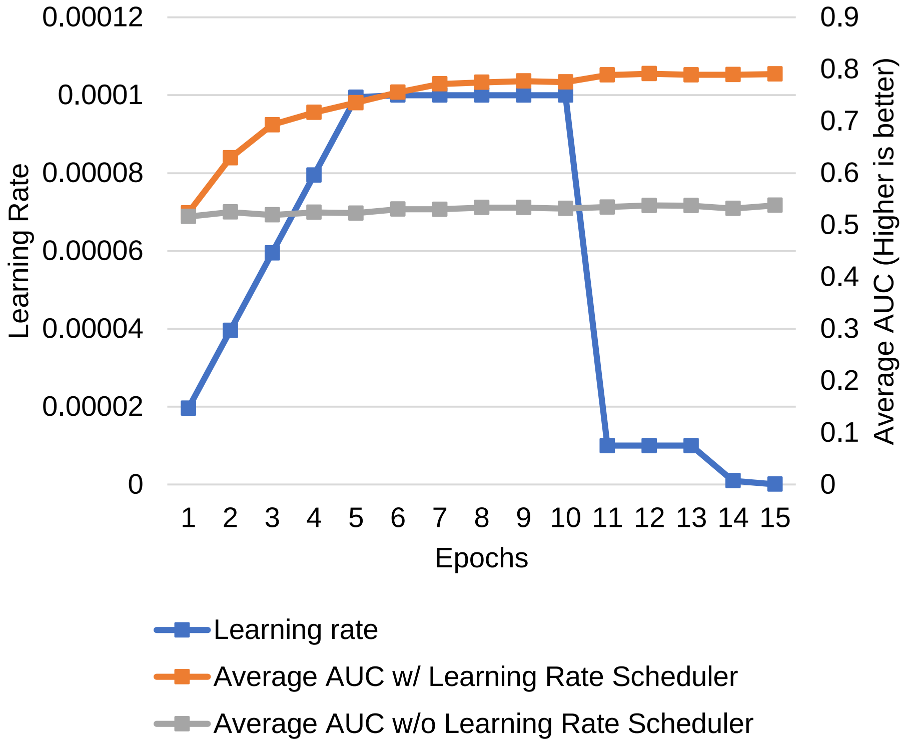
Training the model

- Analytics Zoo API *NNEstimator* to build the model
- *.fit()* produces a neural network model which is a Transformer
- You can now run *.predict()* on the model for inference
- AUC-RoC is used to measure the accuracy of the model. Spark ML pipeline API *BinaryClassificationEvaluator* to determine the AUC-ROC for each disease

```
estimator = NNEstimator(xray_model, BinaryCrossEntropy(), transformer)
             .setBatchSize(batch_size).setMaxEpoch(num_epoch)
            .setFeaturesCol("image").setCachingSample(False).
            .setValidation(EveryEpoch(), validationDF, [AUC()],
batch_size)
            .setOptimMethod(optim_method)
Xray_nnmodel = estimator.fit(trainingDF)
```

Results

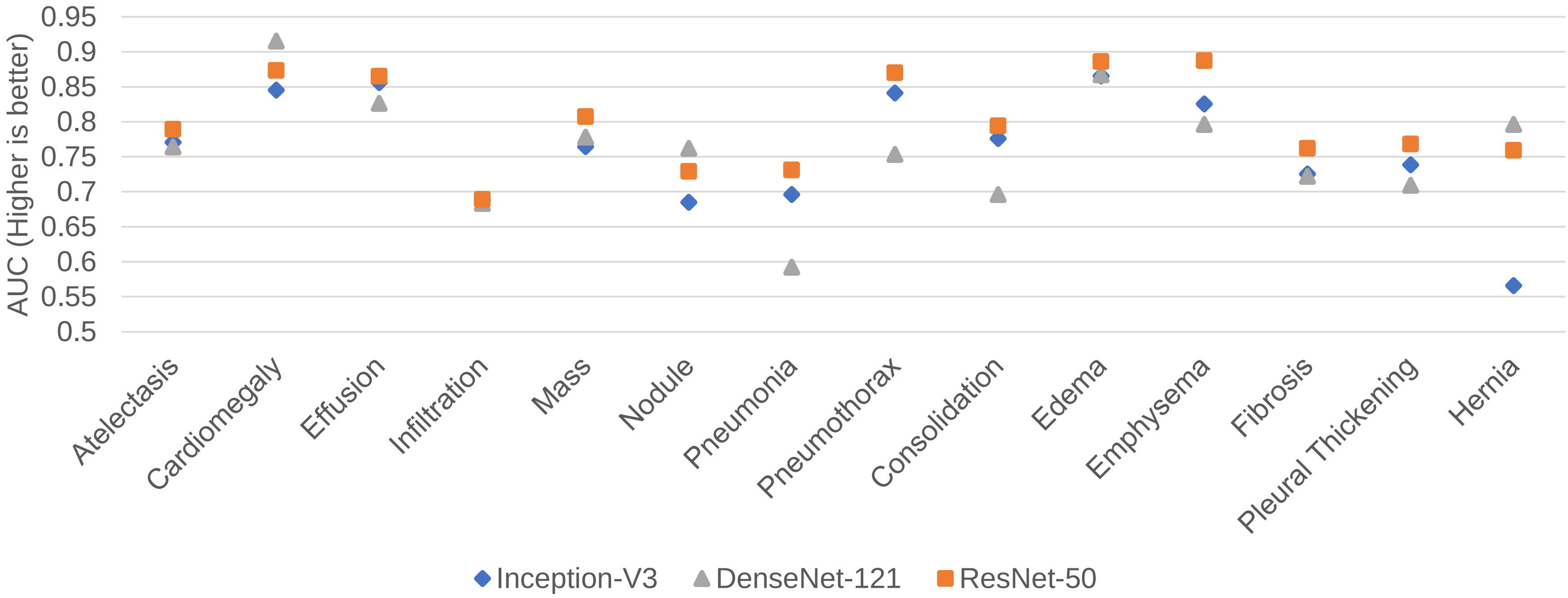
Impact on Learning Rate Scheduler



- Adam (with Learning Rate Scheduler) outperforms SGD
  - Warmup
  - Plateau
- Learning rate scheduler helps is covering the model much faster

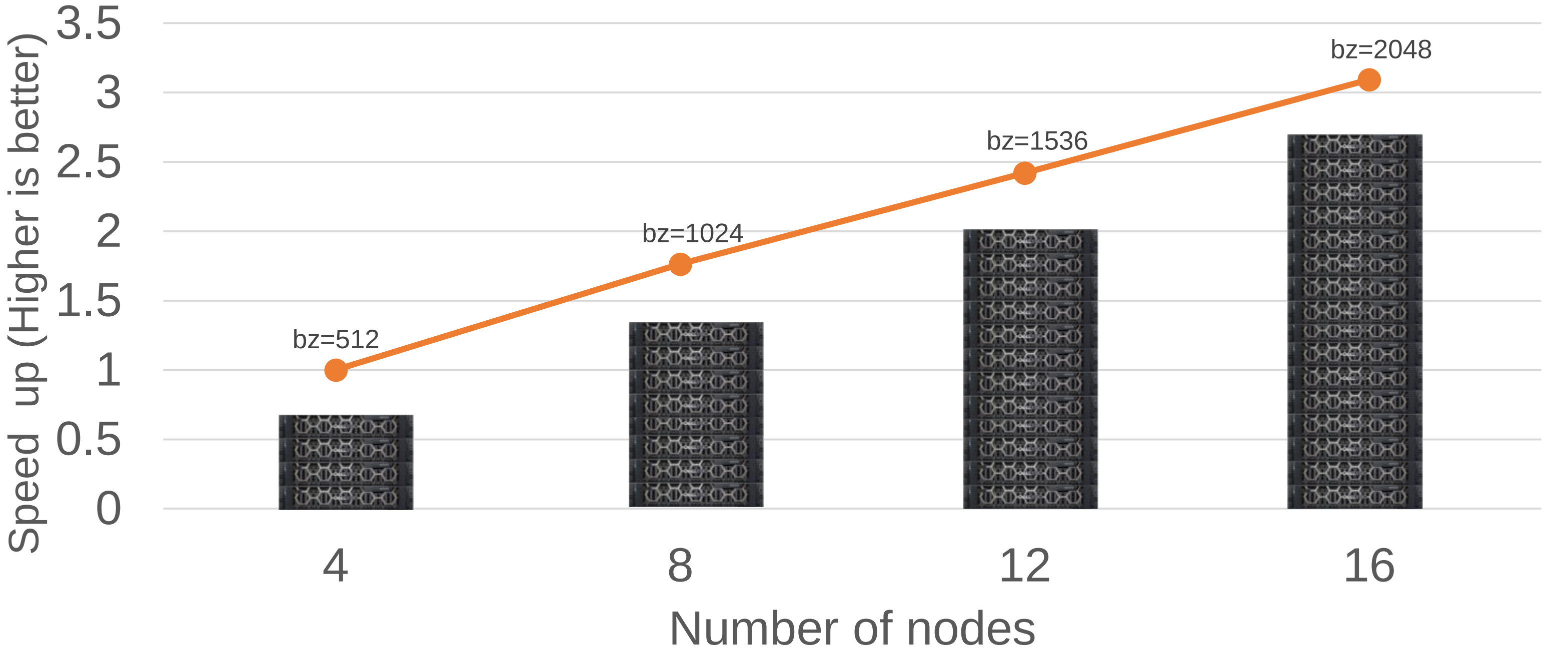
DELL EMC

Base model comparison



Scalability

3x speed up from 4 nodes to 16 nodes.
~ 2.5 hours to train the model



* 15 epochs and 32 cores per executor

Backup

POJO Model Serving API

```
import com.intel.analytics.zoo.pipeline.inference.AbstractInferenceModel;

public class TextClassification extends AbstractInferenceModel {
  public RankerInferenceModel(int concurrentNum) {
    super(concurrentNum);
  }
  ...
}

public class ServingExample {
  public static void main(String[] args) throws IOException {
    TextClassification model = new TextClassification();
    model.load(modelPath, weightPath);

    texts = …
    List<JTensor> inputs = preprocess(texts);
    for (JTensor input : inputs) {
      List<Float> result = model.predict(input.getData(), input.getShape());
      ...
  }
}
```


OpenVINO Support for Model Serving

```
from zoo.common.nncontext import init_nncontext
from zoo.feature.image import ImageSet
from zoo.pipeline.inference import InferenceModel

sc = init_nncontext("OpenVINO Object Detection Inference Example")
images = ImageSet.read(options.img_path, sc,
                       resize_height=600, resize_width=600).get_image().collect()
input_data = np.concatenate([image.reshape((1, 1) + image.shape) for image in images], axis=0)

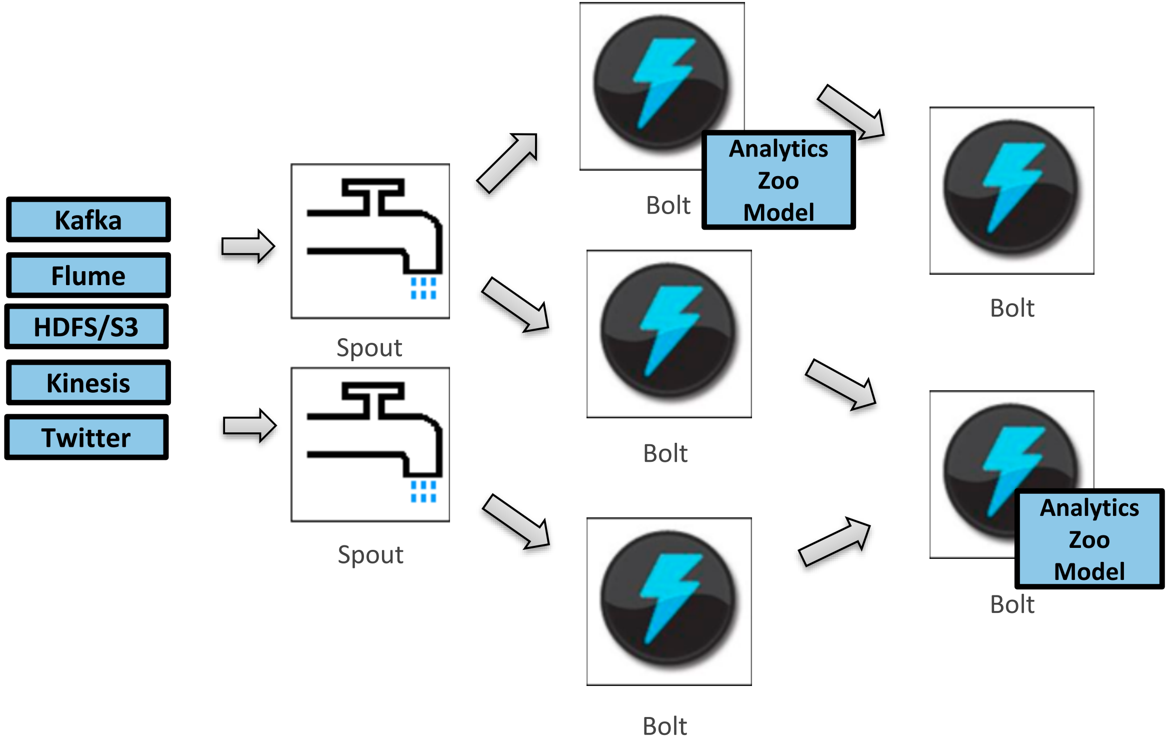
model = InferenceModel()
model.load_tf(options.model_path, backend="openvino", model_type=options.model_type)
predictions = model.predict(input_data)

# Print the detection result of the first image.
print(predictions[0])
```

Transparently support OpenVINO in model serving,
which deliver a significant boost for inference speed

DELLEMC

Model Serving & Inference



Seamless integration in Web Services, Storm, Flink, Kafka, etc. (using POJO *local Java APIs*)