WIFI SSID:SparkAISummit | Password: UnifiedAnalytics

ORGANIZED BY databricks

Using Deep Learning on Apache Spark to diagnose thoracic pathology from chest X-rays

Yuhao Yang, Intel
Bala Chandrasekaran, Dell EMC

#UnifiedAnalytics #SparkAISummit

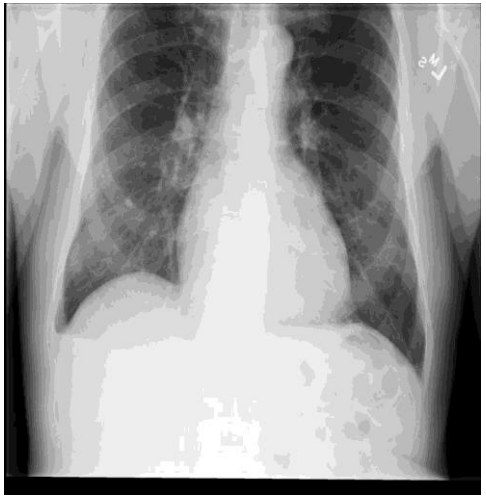
Agenda

- Problem statement
- Analytics Zoo
- Chest Xray Model architecture
- Results and Observations

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/

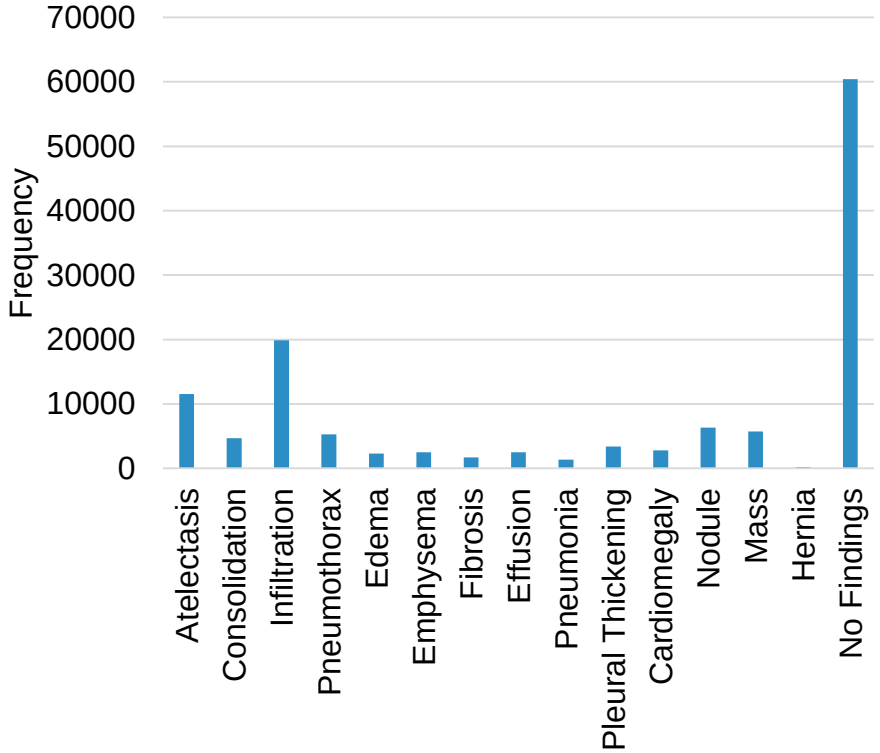


X-ray dataset from NIH

- 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)



Analytics Zoo

Analytics Zoo

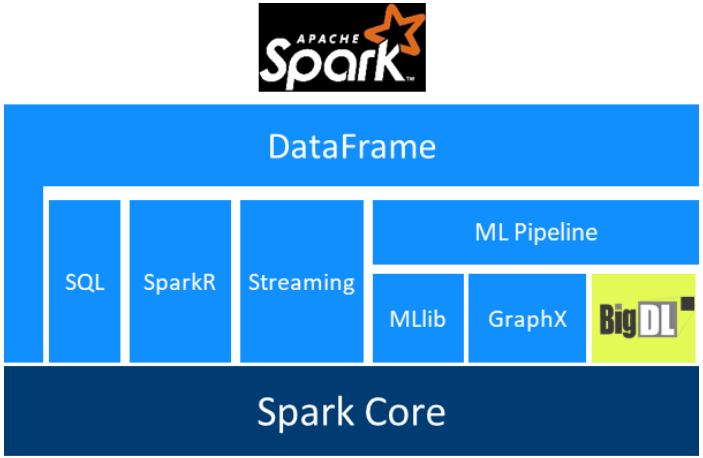
Distributed TensorFlow, Keras and BigDL on Apache Spark

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/

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

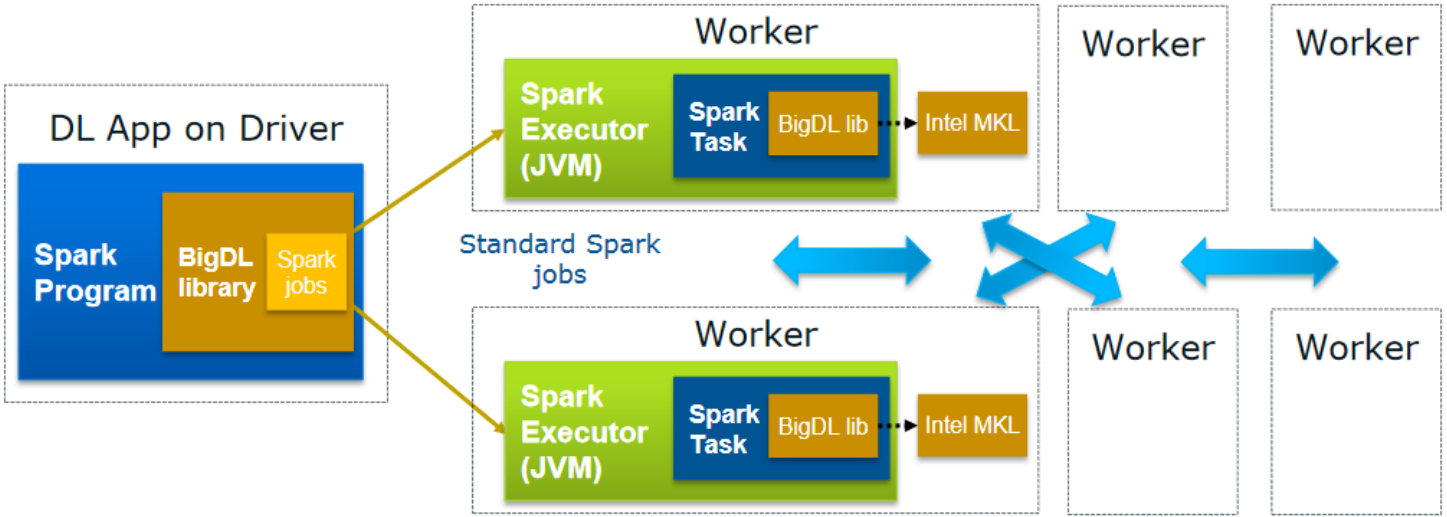


https://github.com/intel-analytics/BigDL

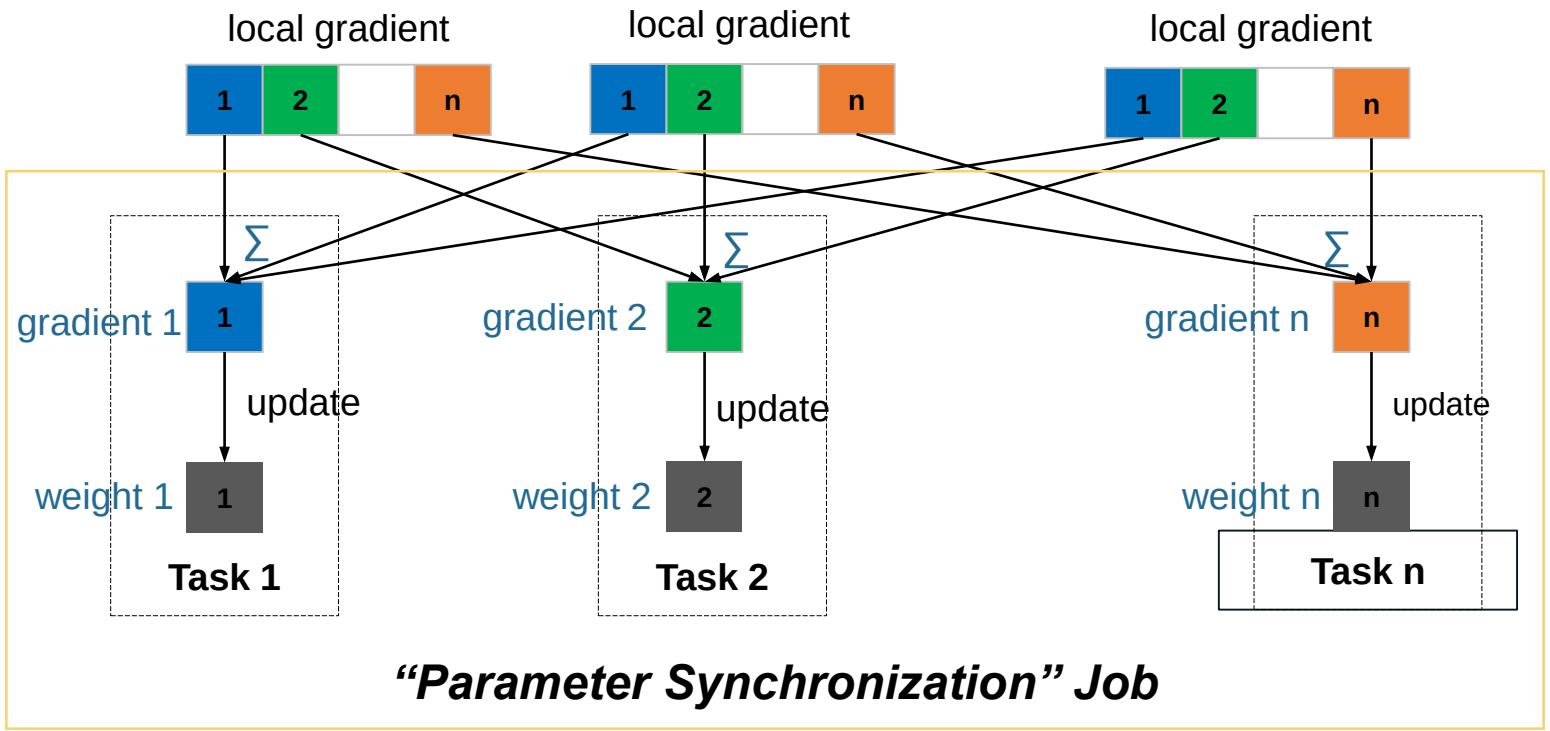
https://bigdl-project.github.io/

Architecture

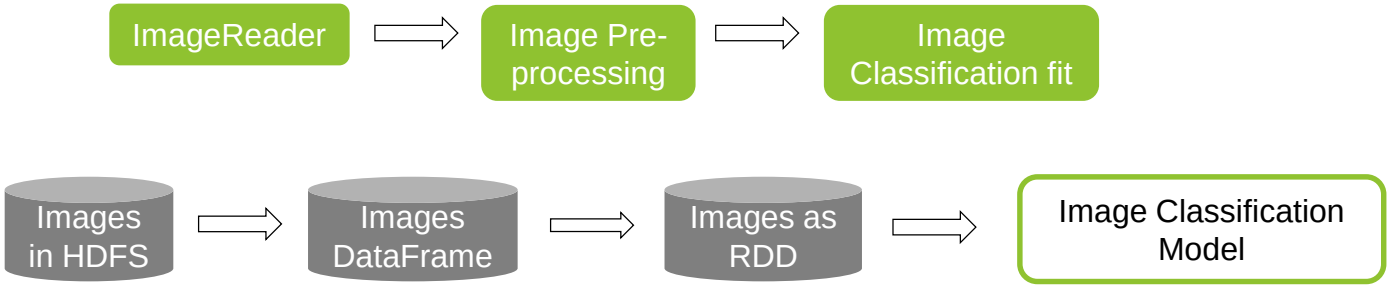
- Training using synchronous mini-batch SGD
- Distributed training iterations run as standard Spark jobs



Parameter Synchronization



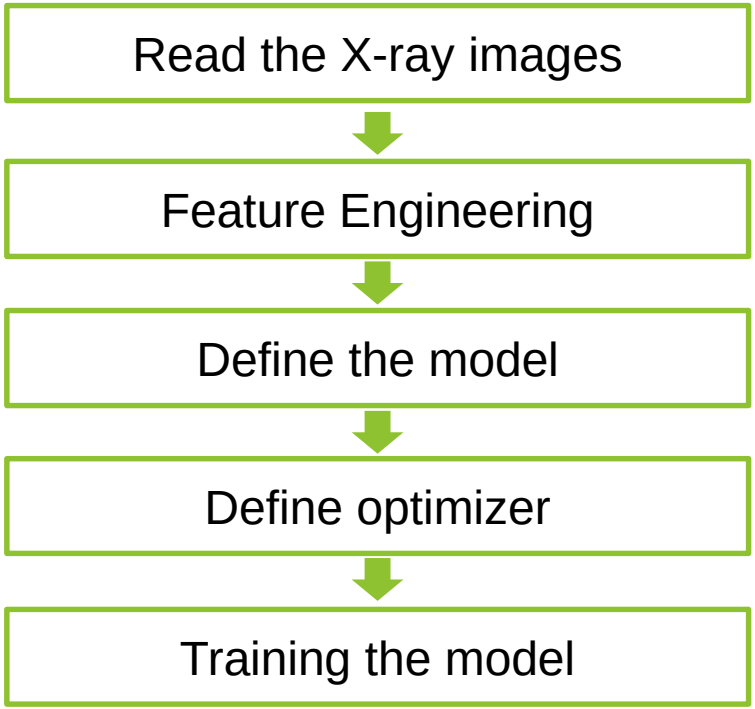
Integrated Pipeline for Machine Learning and Deep Learning



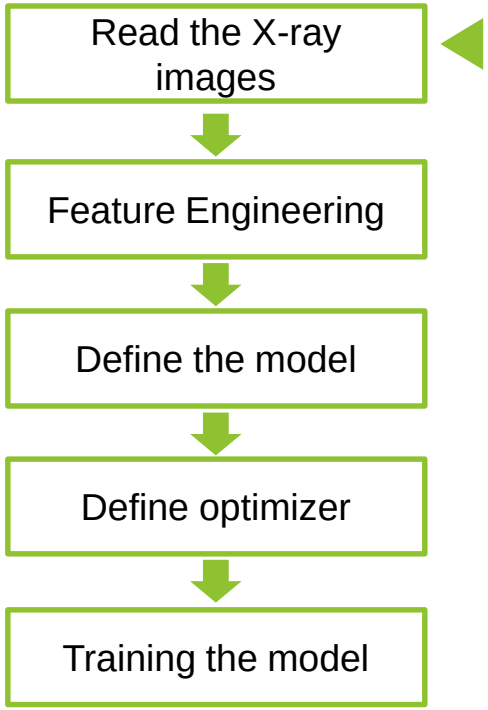
Source: https://spark.apache.org/docs/2.1.1/ml-pipeline.html#estimators

Building the X-ray Model

Let’s build the model



Read the X-ray images as Spark DataFrames



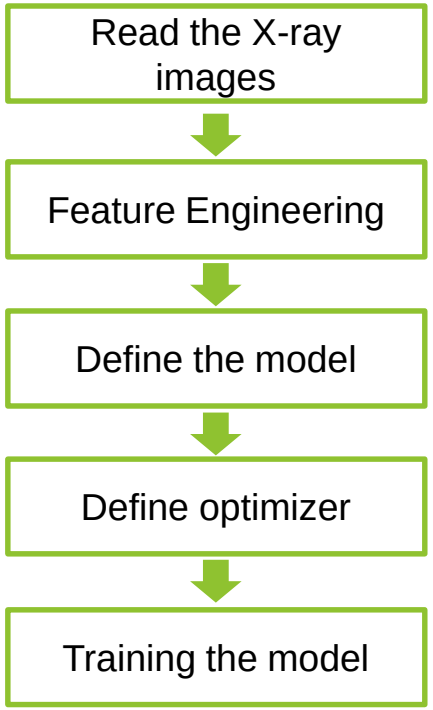
- Initialize *NNContext* and load Xray images into *DataFrames* using *NNImageReader*

```
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



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

ImageCenterCrop(224, 224)

+

Random flip and brightness

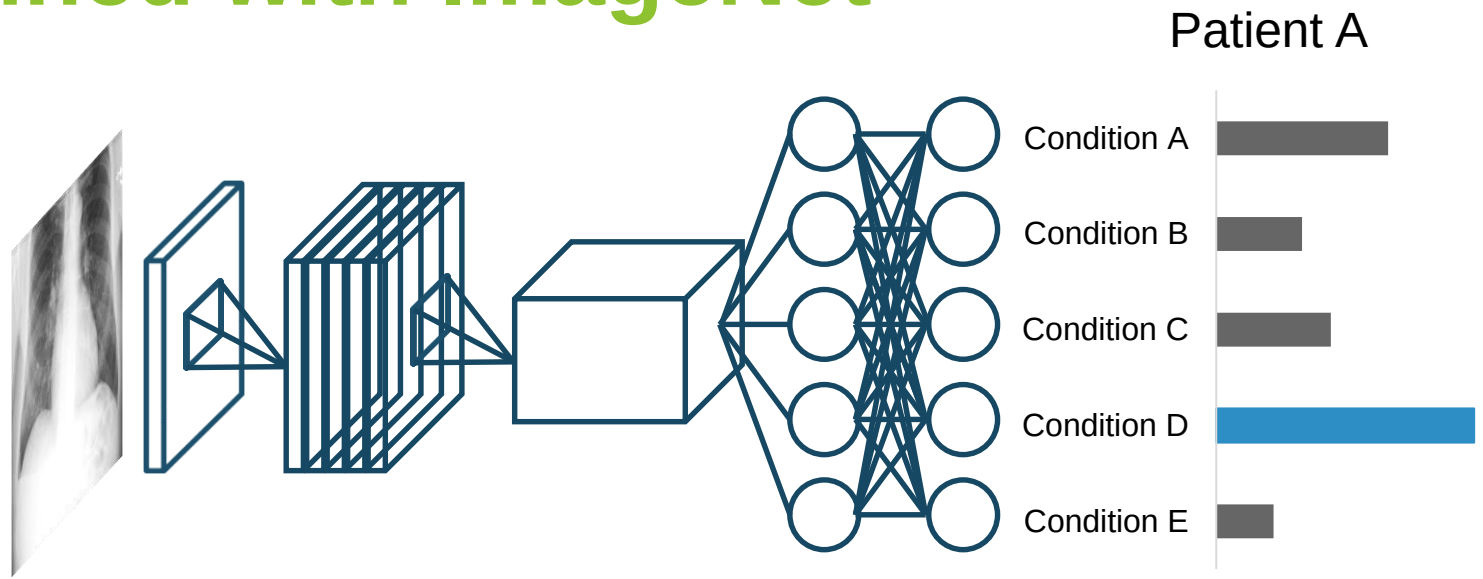
+

ImageChannelNormalize()

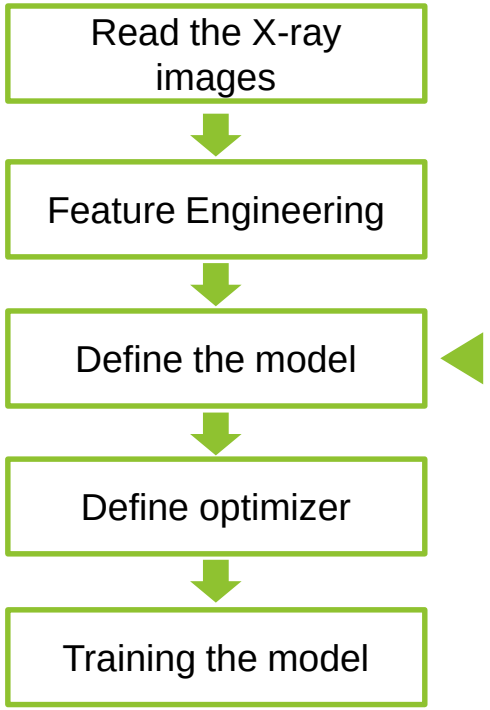
+

To Tensor()

Transfer Learning using ResNet-50 trained with ImageNet

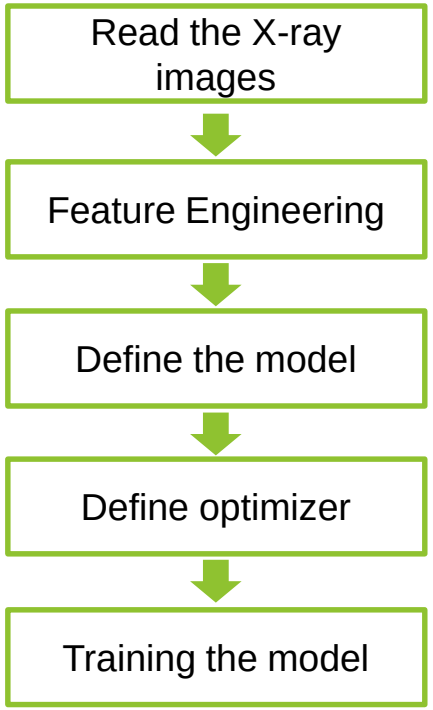


Defining the model with Transfer Learning APIs



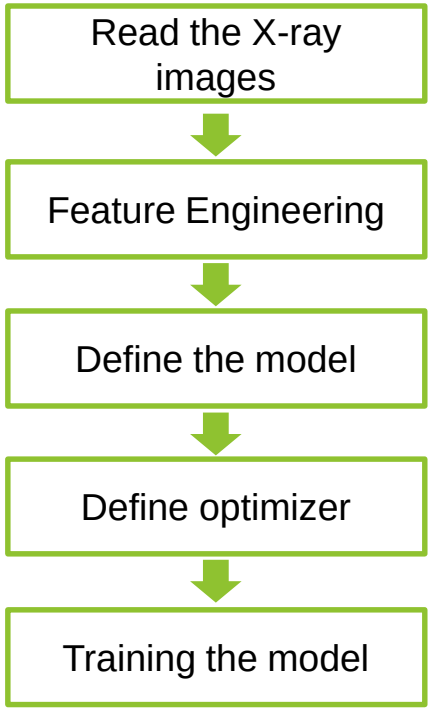
- Load a pre-trained model using *Net.load_bigdl*. The model is trained with ImageNet dataset
  - ResNet 50
  - VGG16
  - DenseNet
  - Inception
- Remove the final *softmax* layer of ResNet-50
- Add new input (for resized x-ray images) and output layer (to predict the 14 diseases). Activation function is *Sigmoid*
- Avoid overfitting
  - Regularization
  - Dropout

Define the Optimizer



- 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)

Train the model using ML Pipelines



- 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)
```

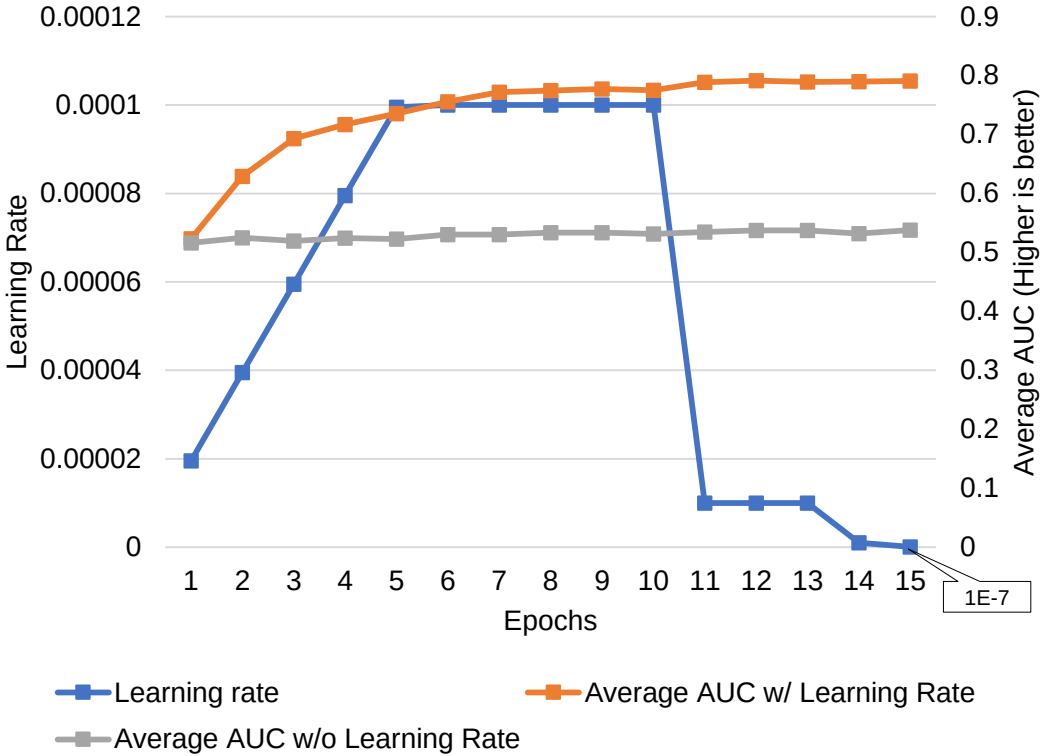
Results and observations

Spark parameter recommendations

- Number of executors ⇨ Number of compute nodes
- Number of executor cores ⇨ Number of physical cores minus two
- Executor memory ⇨

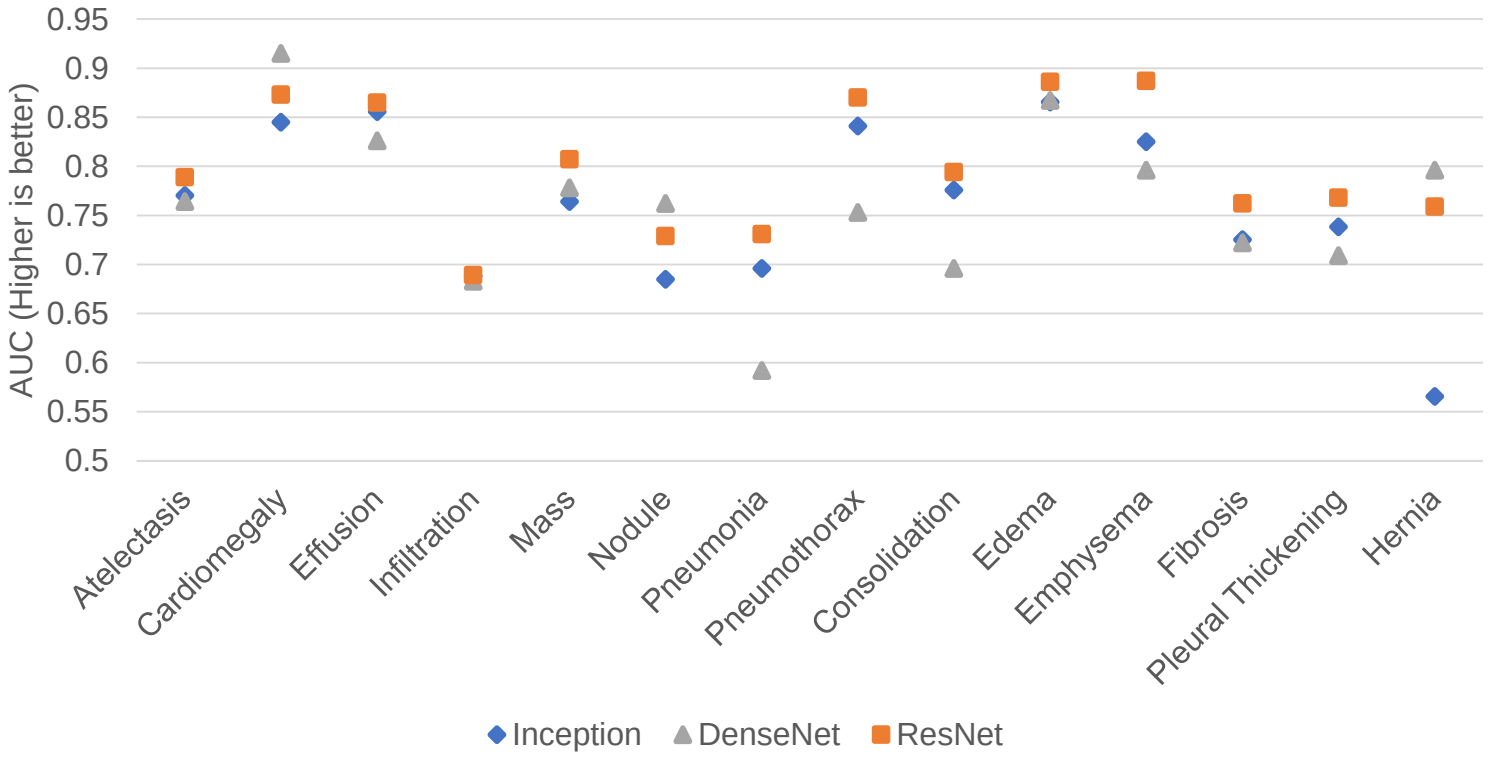
  Number of cores per executor * (Memory required for each core + data partition)

Impact on Learning Rate Scheduler



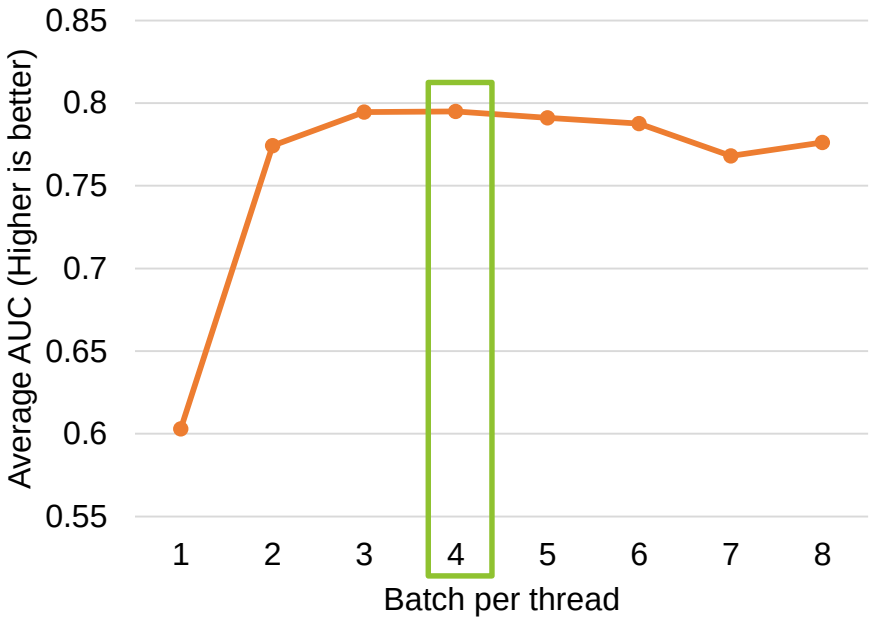
- Adam (with Learning Rate Scheduler) outperforms SGD
- Learning rate scheduler helps is covering the model much faster

Base model comparison



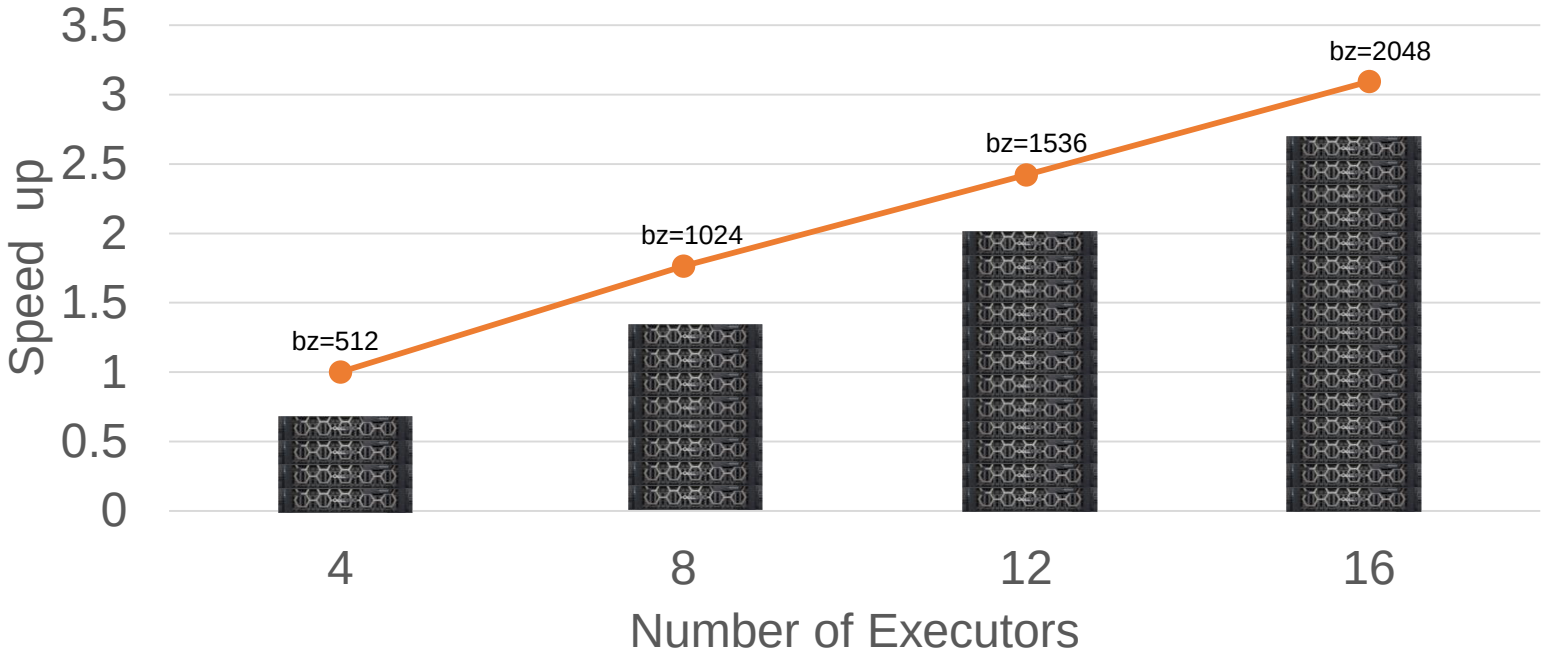
Batch size and Scalability

- 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



Scalability

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



Future Work

- Study the scalability of the model beyond 16 nodes
  - Implement LARS and other advanced learning rate scheduler for large scale training
- Develop performance characteristic and sizing guidelines for image classification problems

Links

Code and white paper available at https://github.com/dell-ai-engineering/BigDL-ImageProcessing-Examples

Acknowledgements

(Alphabetical)

Mehmood Abd, Michael Bennett, Sajan Govindan, Jenwei Hsieh, Andrew Kipp, Dharmesh Patel, Leela Uppuluri, and Luke Wilson

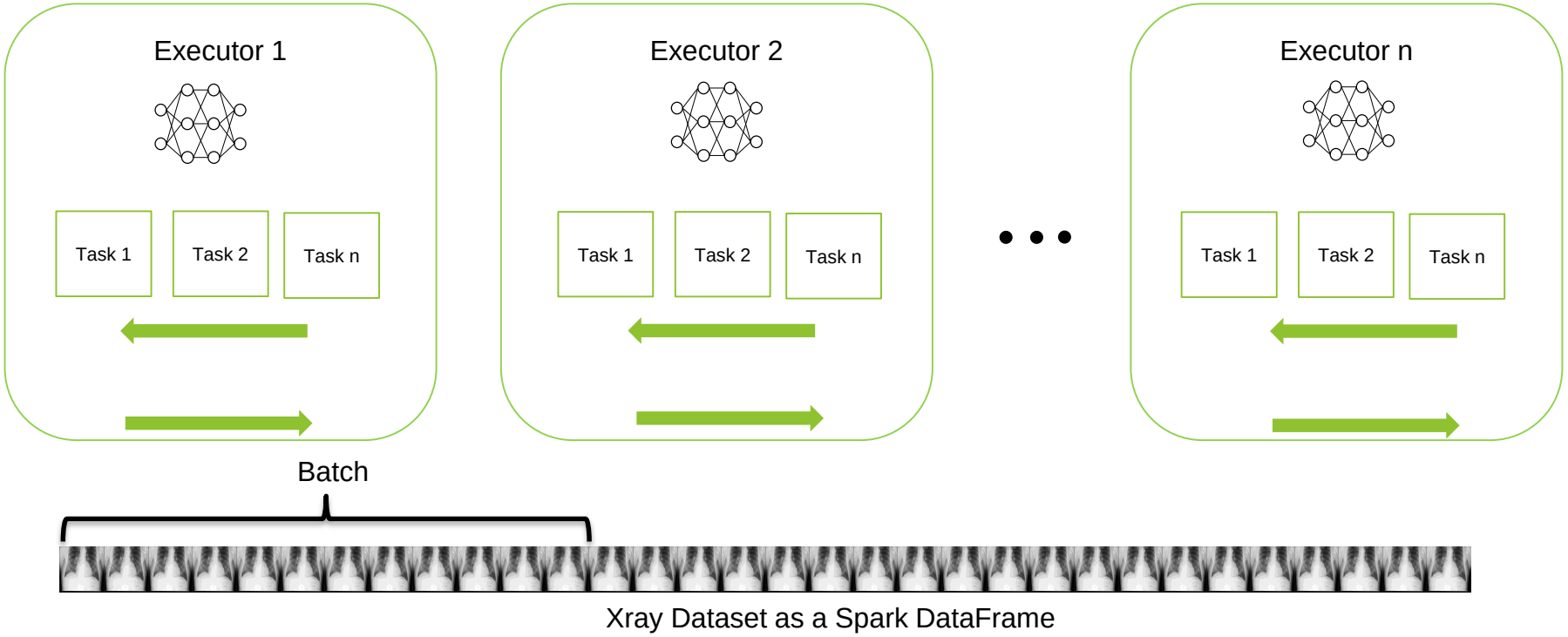
SPARK+AI SUMMIT 2019

DON'T FORGET TO RATE AND REVIEW THE SESSIONS

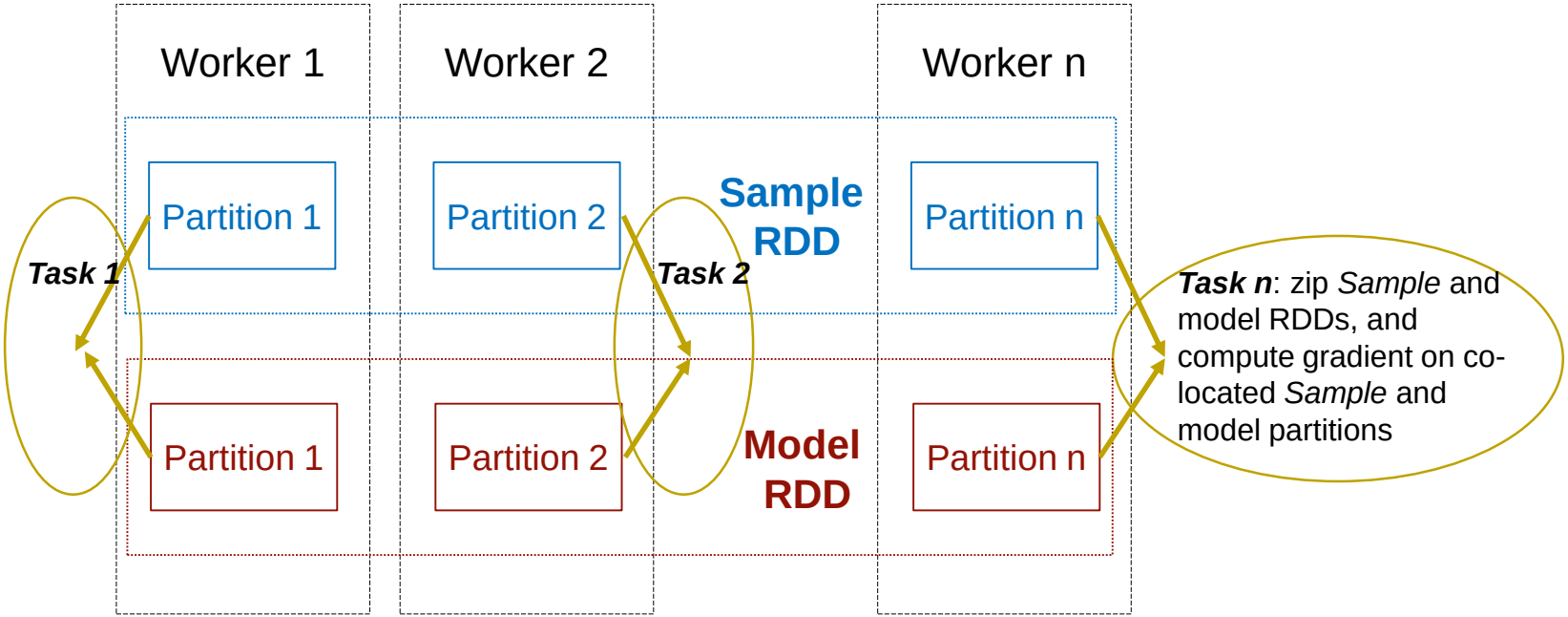
SEARCH SPARK + AI SUMMIT

SPARK+AI

Distributed training in BigDL



Distributed Training in BigDL



Read the X-ray images as Spark DataFrames

- Initialize*NNContext* and load Xray 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

```
getName = udf(lambda row: os.path.basename(row[0]), StringType())
trainingDF = imageDF.join(labelDF, on="Image_Index", how="inner").
                withColumn("Image_Index", getName(col('image')))
```

Feature Engineering – Image Pre-processing

RowToImage()
+
ImageCenterCrop()
+
ImageFlip()
+
ImageChannelNormalize()
+
ImageMatToTensor()

Built in Feature engineering API - *ChainedPreprocessing*

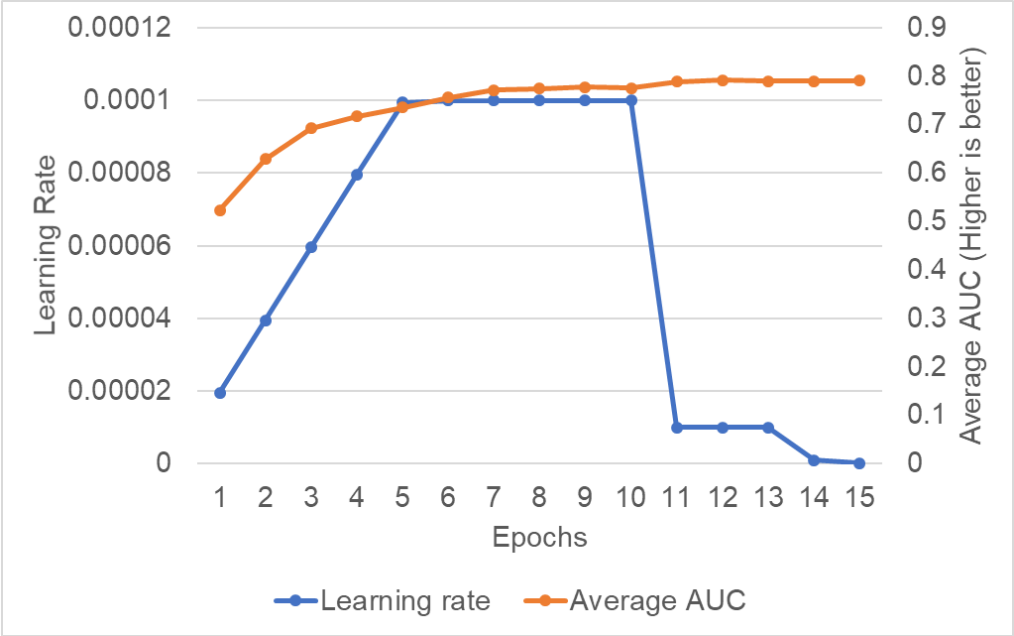
Defining the model with Transfer Learning APIs

- Load a pre-trained model. The model is trained with ImageNet dataset
- Remove the final *softmax* layer of ResNet-50
- Add new input (for resized Xray images) and output layer (to predict the 14 diseases)
- Activation function is *Sigmoid*
- Used regularizations to avoid overfitting

```
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

- Evaluated two optimizers: SGD and Adam Optimizer
  - Adam outperforms SGD
- Learning rate scheduler is implemented in two phases:
  - Warmup: Gradually increase the learning rate for 5 epochs
  - Cool down: Hold the learning rate for a few epochs before cooling down till desired accuracy is achieved



Train the model using ML Pipelines

- 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)
            .setTrainSummary(train_summary).setValidationSummary(val_summary)
            .setOptimMethod(optim_method)
Xray_nnmodel = estimator.fit(trainingDF)
```