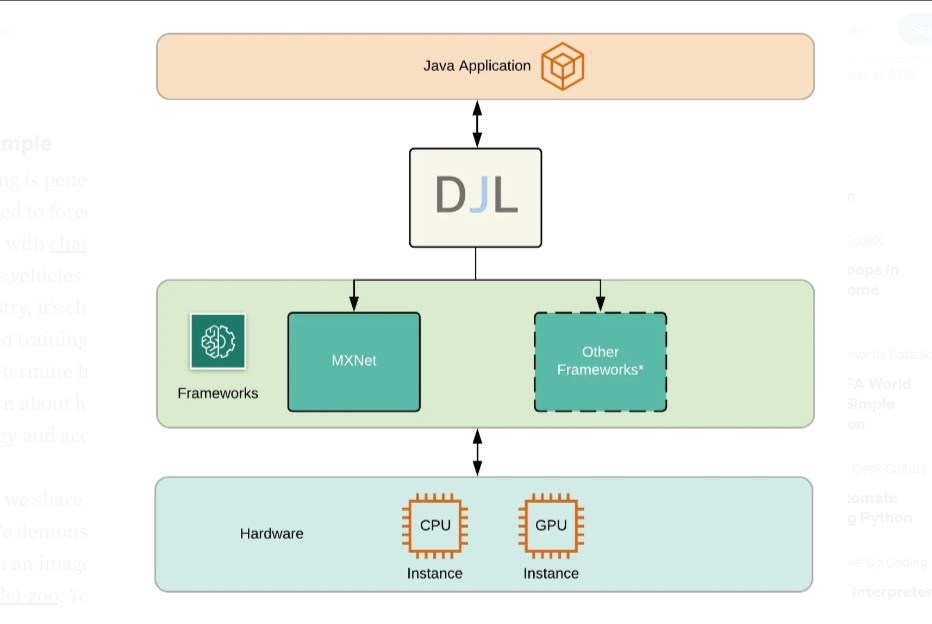
**Key Takeaways:**

* DJL is Developed by Amazon.
* Developers can build, train, and deploy machine learning (ML) and deep learning (DL) models using Java and their favourite IDE.
* DJL simplifies the use of deep learning (DL) frameworks and currently supports Apache MXNet.
* The open-source nature of DJL should be mutually beneficial for the toolkit and its users.
* DJL is engine agnostic, which means developers can write code once and run it on any engine.
* Java developers should have an understanding of the ML lifecycle and common ML terms before attempting to use DJL.
* DJL stays true to Java’s motto, "write once, run anywhere (WORA)", by being engine and deep learning framework-agnostic.
* Developers can write code once that runs on any engine.
* DJL currently provides an implementation for Apache MXNet, an ML engine that eases the development of deep neural networks.
* DJL APIs use JNA, Java Native Access, to call the corresponding Apache MXNet operations.
* DJL orchestrates infrastructure management providing automatic CPU/GPU detection based on the hardware configuration to ensure good performance.

The Deep Java Library (DJL) is an open source library to develop, train and run Deep learning models in Java using intuitive, high-level APIs. If you are a Java user interested in learning Deep learning, DJL is a great way to start learning. If you’re a Java developer working with Deep learning models, DJL will simplify the way you train and run predictions. In this Article, we will show how to run a prediction with a pre-trained Deep learning model.



**DJL Example:**

Deep learning is penetrating into enterprise across a variety of use cases. In retail, it’s used to forecast customer demand and analyze customer interactions with chatbots.

In the automotive industry, it is used to navigate autonomous vehicles and find quality defects in manufacturing. and

In the sports industry, it’s changing the way the game is played with real-time coaching and training insights. Imagine being able to model your opponents moves or determine how to position your team using deep learning models.

You can learn about how to Identify Objects from an Image in this article.

**Dependencies/Engine’s:**

<!-- https://mvnrepository.com/artifact/ai.djl/api -->

<dependency>

<groupId>ai.djl</groupId>

<artifactId>api</artifactId>

<version>0.20.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/ai.djl/basicdataset -->

<dependency>

<groupId>ai.djl</groupId>

<artifactId>basicdataset</artifactId>

<version>0.20.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/ai.djl.timeseries/timeseries -->

<dependency>

<groupId>ai.djl.timeseries</groupId>

<artifactId>timeseries</artifactId>

<version>0.20.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/ai.djl.mxnet/mxnet-model-zoo -->

<dependency>

<groupId>ai.djl.mxnet</groupId>

<artifactId>mxnet-model-zoo</artifactId>

<version>0.19.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/ai.djl.pytorch/pytorch-model-zoo -->

<dependency>

<groupId>ai.djl.pytorch</groupId>

<artifactId>pytorch-model-zoo</artifactId>

<version>0.19.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/ai.djl.pytorch/pytorch-engine -->

<dependency>

<groupId>ai.djl.pytorch</groupId>

<artifactId>pytorch-engine</artifactId>

<version>0.19.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.tensorflow/tensorflow -->

<dependency>

<groupId>org.tensorflow</groupId>

<artifactId>tensorflow</artifactId>

<version>1.15.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact /paddlepaddle-model-zoo -->

<dependency>

<groupId>ai.djl.paddlepaddle</groupId>

<artifactId>paddlepaddle-model-zoo</artifactId>

<version>0.19.0</version>

</dependency>

<dependency>

<groupId>ai.djl.paddlepaddle</groupId>

<artifactId>paddlepaddle-engine</artifactId>

<version>0.19.0</version>

<scope>runtime</scope>

</dependency>

<dependency>

<groupId>ai.djl.tensorflow</groupId>

<artifactId>tensorflow-model-zoo</artifactId>

<version>0.20.0</version>

</dependency>

<dependency>

<groupId>commons-cli</groupId>

<artifactId>commons-cli</artifactId>

<version>1.5.0</version>

</dependency>

<dependency>

<groupId>org.apache.logging.log4j</groupId>

<artifactId>log4j-slf4j-impl</artifactId>

<version>2.18.0</version>

</dependency>

<!-- https://mvnrepository.com/artifact/org.testng/testng -->

<dependency>

<groupId>org.testng</groupId>

<artifactId>testng</artifactId>

<version>7.5</version>

<scope>test</scope>

</dependency>

**Steps to Create Project:**

Step:1 Create Simple Maven Project into Eclipse or STS

Step:2 Open the pom.xml and add Dependencies with Engine.

Step:3 Clone my Repository

Step:4 Copy my code into project

Step:5 Run the Project and here we got the Result like this…



This is Input Image

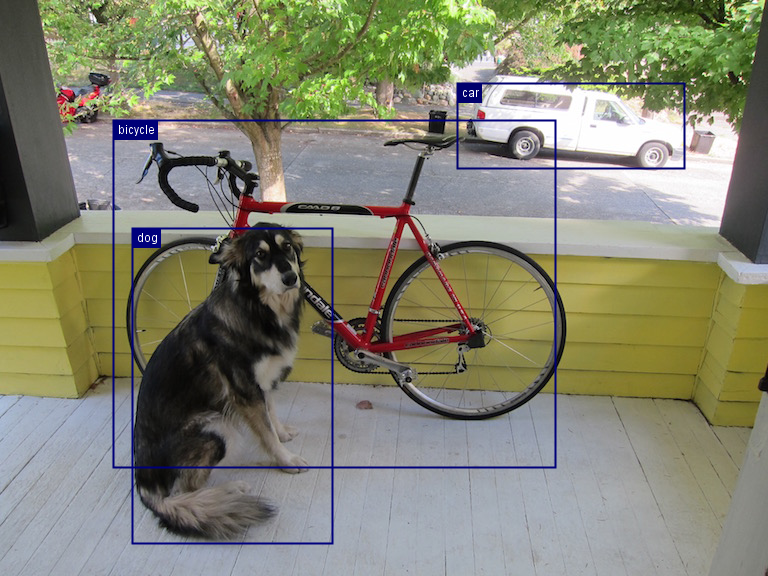


This is Our First Image Result/Output.

After Changing Input Image.



Now this is our Input Image.



Now this will be the Output for Second Input Image.

**Explanation:**

Here First we create our simple maven project and After that we add some dependencies and engine from mvn repository official site. After this all we need to code.

First we create two class first which contain our main class and second which contains our engine which is TensorFlow.

This below code will run means we start prediction on our input image by this block of code.

**public** **static** DetectedObjects predict() **throws** IOException, ModelException, TranslateException

{

Path imageFile = Paths.*get*("src/test/input\_img/dog\_bike\_car.jpg");

//for image detection

Image img = ImageFactory.*getInstance*().fromFile(imageFile);

String modelUrl =

"http://download.tensorflow.org/models/object\_detection/tf2/20200711/ssd\_mobilenet\_v2\_320x320\_coco17\_tpu-8.tar.gz";

Criteria<Image, DetectedObjects> criteria =

Criteria.*builder*()

.optApplication(Application.CV.***OBJECT\_DETECTION***)

.setTypes(Image.**class**, DetectedObjects.**class**)

.optModelUrls(modelUrl)

.optModelName("saved\_model")

.optTranslator(**new** MyTranslator())

.optEngine("TensorFlow")

.optProgress(**new** ProgressBar())

.build();

**try** (ZooModel<Image, DetectedObjects> model = criteria.loadModel();

Predictor<Image, DetectedObjects> predictor = model.newPredictor())

{

DetectedObjects detection = predictor.predict(img);

*saveBoundingBoxImage*(img, detection);

**return** detection;

}

}