DO

Machine Learning Yourself with Python

Ahmed Fawzy Gad

Do Machine Learning Yourself with Python
Ahmed Fawzy Gad

This book is for sale at http://leanpub.com/doml

This is a Leanpub book. Leanpub empowers authors and publishers with the Lean Publishing process. Lean Publishing is the act of publishing an in-progress e-book using lightweight tools and
many iterations to get reader feedback, pivot until you have the right book and build traction once
you do.

Preface

Do Machine Learning Yourself is a collection of do it yourself (DIY) projects about machine learning, mainly about computer vision, for beginner and intermediate levels. Through a detailed guidance per each project, everything required to do that project yourself will be clear. One primary focus of the book is to make machine learning available for mobile devices.

The book is organized into 8 chapters where each chapter handles a topic. The first chapter is about doing some machine learning projects using OpenCV. Because OpenCV is a cross-platform library, then the projects can run in both desktop and mobile devices. For such a reason, the book builds the project for running in Android devices.

The second chapter builds 3 computer vision projects for Raspberry Pi. This include preparing Raspberry Pi for first time use in order to be able to access it and install some Python libraries. A USB camera is connected to the Raspberry Pi to capture images for building a surveillance system and also a simple robot car.

Chapter 3 discusses some topics about neural network such as selecting the best architecture that works for a given data. Also a neural network is created in Keras that is deployed to a Flask server. An Android client uploads images to be classified by the trained Keras model and then return the response.

Chapter 4 uses the genetic algorithm for building doing some optimization. This include optimizing the 8 queen puzzle to reach a solution. Besides that project, the algorithm is used for reproducing images by evolving the pixel values.

Chapter 5 uses a framework named Kivy for building GUI Python applications. Based on the Kivy app, cross-platform applications can be generated. This book focuses on building Android applications by the help of 2 projects named Python-4-Android and Buildozer. Kivy allows some Python libraries to be executed in Android such as NumPy which is one of the most important libraries for building data science applications. An image classifier is created in Python that is able to work in Android using NumPy. The book also discusses supporting the Arabic text with Kivy.

Chapter 6 uses OpenCV for building an Android application able to do some image effects. The effects supported are horizontal and vertical image stitching, reducing the number of colors representing the both color and gray images by editing the lookup tables, noise removal using the median filter, converting images to binary, cartooning images, image blending using the alpha channel, creating animated GIF image from a number of images, and creating image patterns.

Chapter 7 builds a chat application for Android devices for sending and receiving text messages. The app allows the users to register themselves by entering their information that include a username, password, and an e-mail. One the e-mail address is verified, the user is registered and able to send and receive messages. The project starts by building the database and the tables for holding the users data and also the messages. A Flask-based server is created that is connected to the Android app. The project supports instant notifications for new messages and encryption.

Chapter 8 lists some miscellaneous topics about machine learning. This includes a discussion about how machine learning is not killed by the appearance of deep learning. Also this chapter guides beginners to understand how ensemble algorithms work and understand the difference between bagging and boosting ensemble models. The gradient boosting algorithm is discussed which is an application of ensemble boosting.

It is worth mentioning that the book is aggregating the tutorials and articles I posted in a number of blogs including KDnuggets (https://www.kdnuggets.com/author/ahmed-gad), Heartbeat (https://heartbeat.fritz.ai/@ahmedfgad), Towards Data Science (https://towardsdatascience.com/@ahmedfgad), and Medium (https://medium.com/@ahmedfgad). The book organizes these tutorials so that one tutorial takes you to another one in a way that let you feel they are a single part. The tutorials are published in the last 2 years and new ones will be available in the future to give a guide about some topics about software engineering, machine learning, deep learning, computer vision, and Python.

Acknowledgment

First of all, I have to thanks Allah for granting me some of his knowledge and helping me to write tens of tutorials and preparing 3 books up to 2019. I did not realize that I can do all of this stuff but nothing is impossible.

I remember that day by the end of 2017 when I wrote my first tutorial about building artificial neural networks using TensorFlow. I put KDnuggets as a target as it is an achievement to publish there. I contacted Matthew Mayo, researcher and editor at KDnuggets, and he accepted it. He is someone you like to know and work with. I admit he played a crucial rule in making me visible in the community.

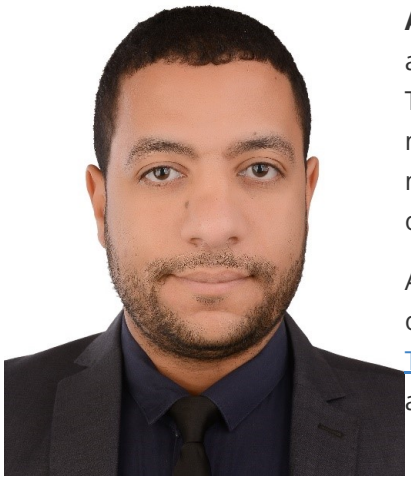
Later, I started contributing to Heartbeat, a Fritz AI blog lead by Austin Kodra and Jameson Toole. They target reducing the gap between machine learning mobile devices. I enjoyed their target and frequently build projects that run in Android and Raspberry Pi and document them in tutorials. Austin, the head of the community, offers all help to increase the quality of the tutorials and with his critical feedback. He is a consistent hard-working man and I am sure Fritz AI will be a known name for all people in the field.

I also started with Paperspace, a cloud-computing platform that helps to build machine learning and deep learning applications. The CEO of Paperspace Dillon Erb and its COO Daniel Kobran welcome the idea and I started contributing with my writings there. They offer the necessary tools for building projects based on their product called Gradient that provides effortless tools for building and deploying machine learning models. The blog editor, Rachel Rapp, always do the necessary revisions and edits over my writings until making it ready for the public.

As usual, some people play an important rule in encouraging and supporting me to go further and Dr. Mahmoud Albawaneh, Executive Director of Institutional Research & Analytics at California State University, Long Beach, is at the top. He never saves effort in offering help and support in a way that makes me feel calm and optimistic for the future.

Besides the revision offered by the blog editors, I always receive feedback over my writings from Fatima Ezzahra Jarmouni which is an M.Sc. data scientist. She never keeps efforts to offer her help as she reviews my posts and responds with some high-quality opinions.

About the Author



Ahmed Fawzy Gad is a machine learning engineer teaching assistant who received his B.Sc. and M.Sc. in Information Technology. Ahmed is a software learning engineer interested in machine/deep learning, computer vision, and Python. He is a machine learning technical reviewer and consultant helping others do their projects.

Ahmed contributes by written tutorials and articles to a number of blogs including Paperspace, KDnuggets, Heartbeat, and Towards Data Science. He has more than 60 tutorials and articles.

Ahmed authored 3 technical books from 2017 to 2019 which are:

1. TensorFlow: A Guide to Build Artificial Neural Networks using Python (Labmert 2017)
2. Practical Computer Vision Applications Using Deep Learning with CNNs (Apress, 2018)
3. Building Android Apps in Python Using Kivy with Android Studio (Apress, 2019)

Ahmed is enthusiastic to start his Ph.D. degree in machine learning and looking for new work positions. He welcomes connecting with him through

- LinkedIn (https://www.linkedin.com/in/ahmedfgad)
- Twitter (https://twitter.com/ahmedfgad)
- Facebook (https://www.facebook.com/ahmed.f.gadd)
- E-mail (ahmed.f.gad@gmail.com)

Chapter 1: Machine Learning with OpenCV

The sample pages just include the second section of Chapter 1.

Running Artificial Neural Networks in Android using OpenCV

A step-by-step guide will be given for building an artificial neural network (ANN) using OpenCV for Android devices. An ANN for creating a 2-input XOR gate is built and trained on a desktop computer and then saved for later use in an Android app.

The IDE used for building the desktop app is NetBeans and the OS is Windows. For detailed steps covering preparing OpenCV for Android, you can read section 1.1 titled **A Guide to Preparing OpenCV for Android**.

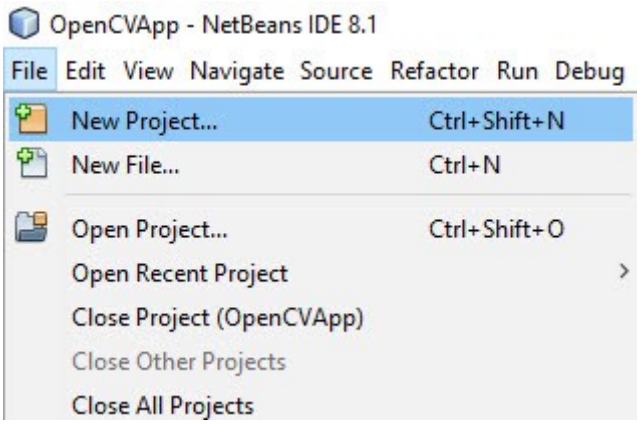
You can also avoid preparing OpenCV for Android from scratch by downloading the Android Studio project in which OpenCV is already linked. The project is available on GitHub:

https://github.com/ahmedfgad/OpenCVAndroid

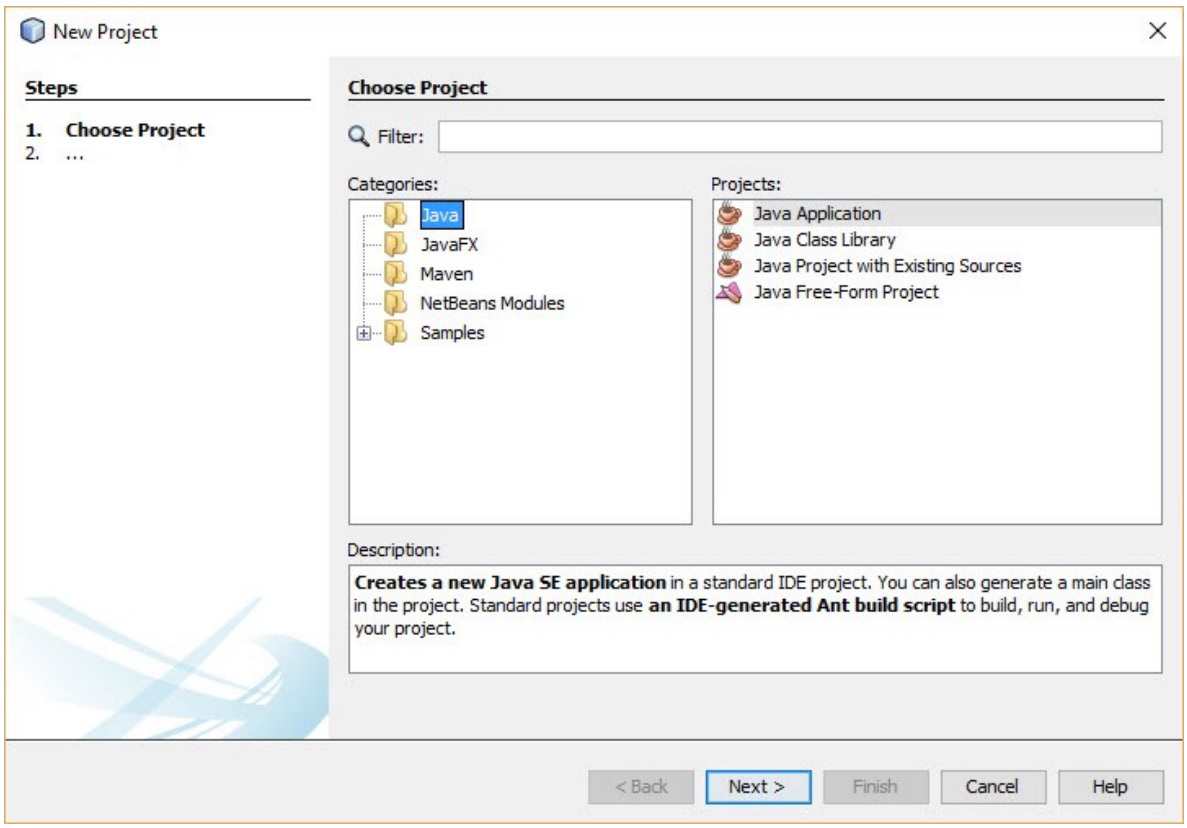
Creating a New Java Application in NetBeans

The first step is to create a new Java app in the IDE we are using which is NetBeans. If you don't have this IDE, you can download it here: https://netbeans.org/downloads/8.0.2.

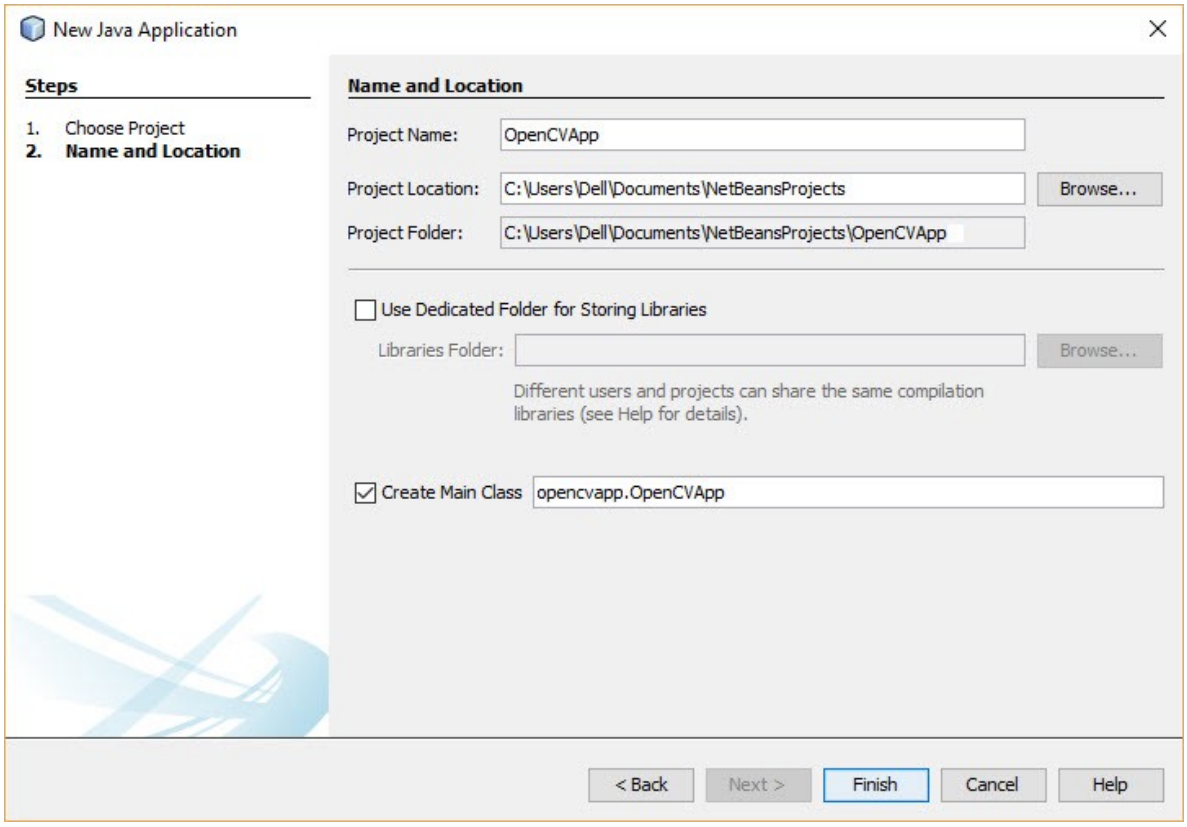
After downloading and installing the IDE, open it and click on the **File** menu and select **New Project**:



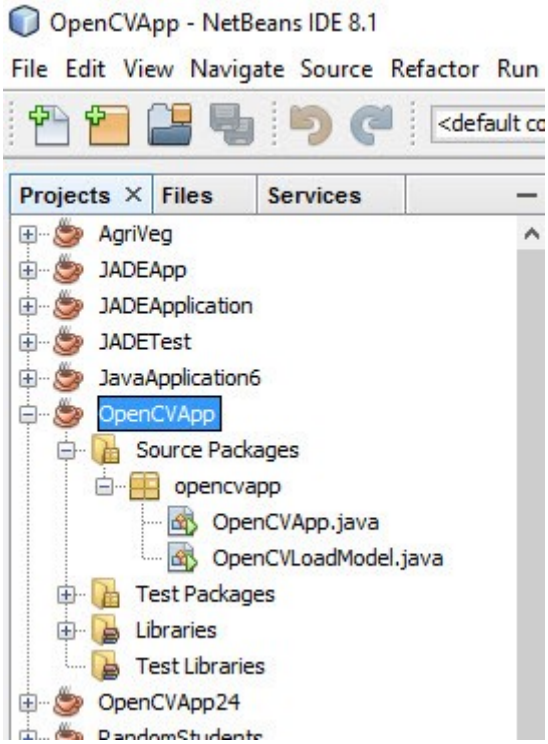
This opens a new window as shown below for selecting the project type. From the **Categories** list, click on **Java**. From the **Projects** list, select **Java Application**.



After clicking on the **Next** button, a new window opens for entering the project name and path. Enter the details of your choice. For me, the project name is set to **OpenCVApp**, and thus the package name is automatically set to **opencvapp**. I prefer checking the "**Create Main Class**" checkbox [which is already checked by default] for creating a default class within the package. As specified on the right of the checkbox, the class name is **OpenCVApp**, which resided within the **opencvapp** package. Click on Finish to build the project.



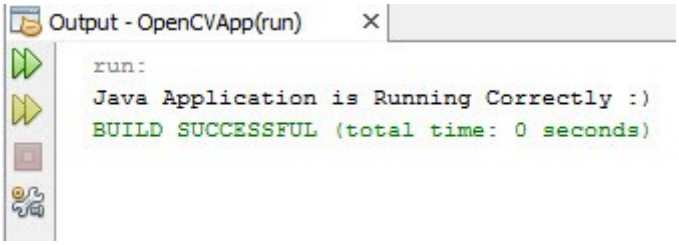
The project structure is shown in the next figure. Note that I created a new class named `OpenCVLoadModel` that will be used later for loading the trained ANN model. You can either create it now or later once the trained model is created.



The `OpenCVApp` class has just an `empty main()` method. We can print a message to make sure the project is working correctly. The implementation of this class is as shown below, in which a print message displays `Java Application is Running Correctly :)` once the class runs.

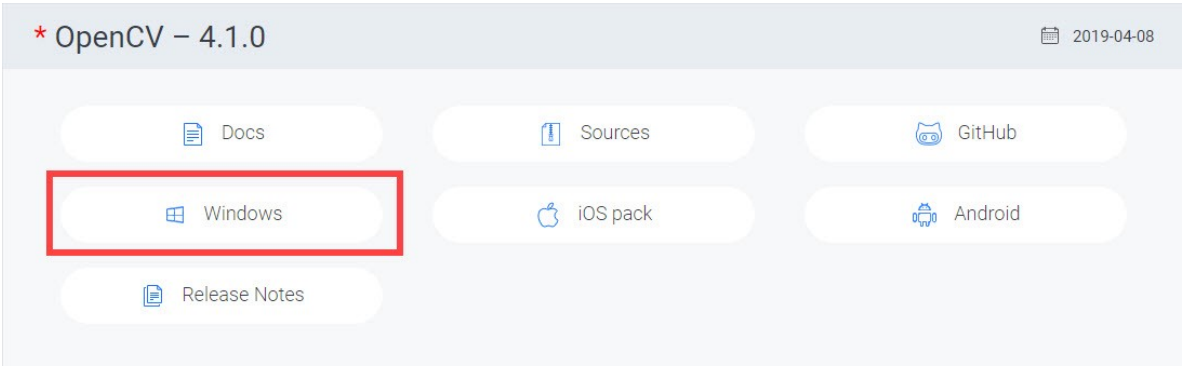
```
package opencvapp;
public class OpenCVApp{
public static void main(String[] args) {
    System.out.println("Java Application is Running Correctly :) ");
}
}
```

The console output of the run is shown in the next figure. After making sure the project is working as expected, the next step is to download OpenCV for Windows and import it within NetBeans.



Downloading OpenCV for Windows

By visiting the OpenCV releases download page, you can find all OpenCV releases available for different distributions. Let's use the latest OpenCV release, which is 4.1.0. Because we're using Windows, the Windows distribution will be downloaded. This will download an .exe file which you can extract as regular compressed files.

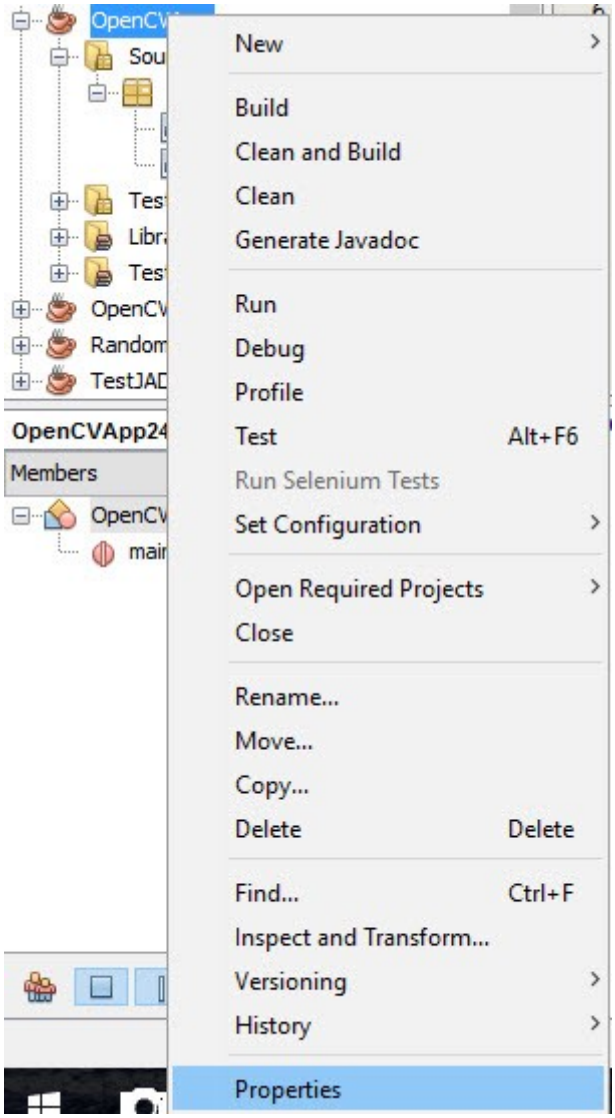


Locate the extracted OpenCV folder and go to `\opencv\build\java`. Within this folder, you will find the contents listed below. The first 2 folders contain a DLL file, which will be used in the next step. The JAR file contains the OpenCV classes.

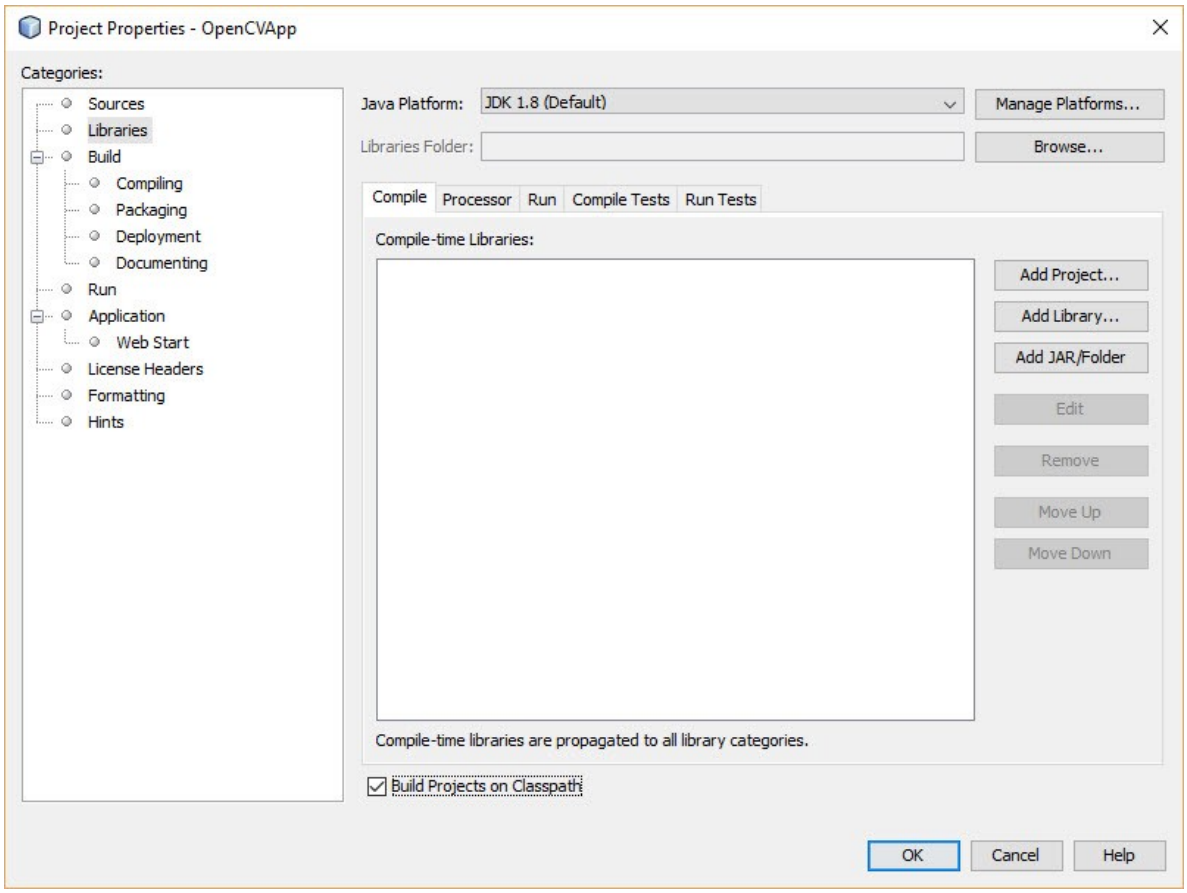
- x64/
- x86/
- opencv-410.jar

Importing OpenCV in NetBeans

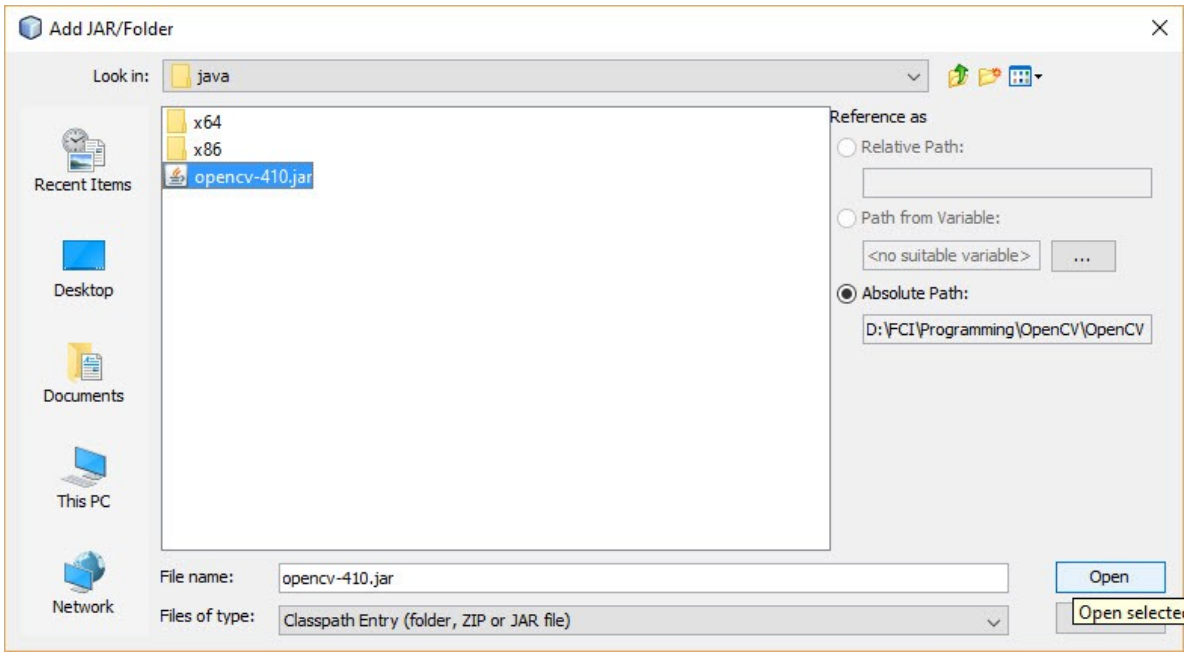
The JAR file (**opencv-410.jar**) must be imported within the project in order to use OpenCV. In order to import a JAR file within NetBeans, right click on the project and select **Properties** as illustrated below.



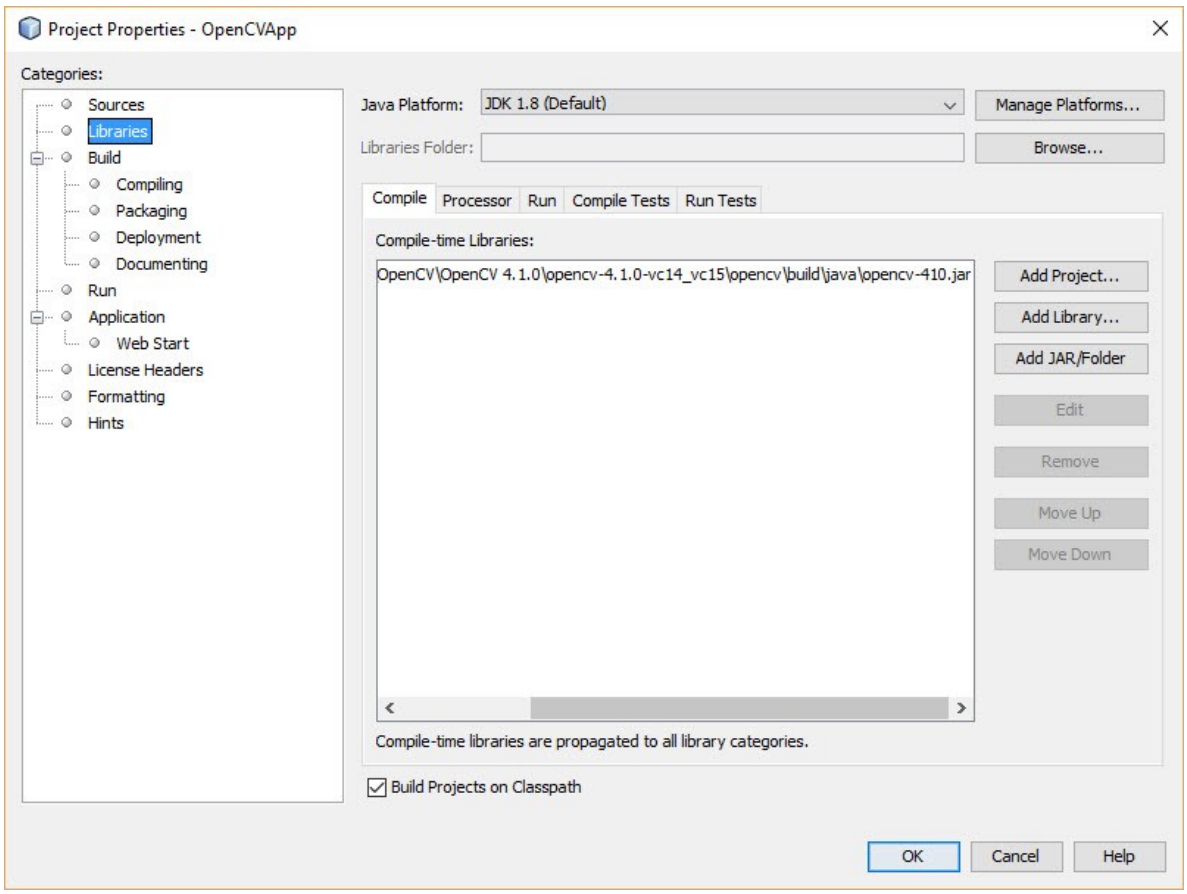
This opens the project properties window shown next. The window does not yet list any libraries used by the project.



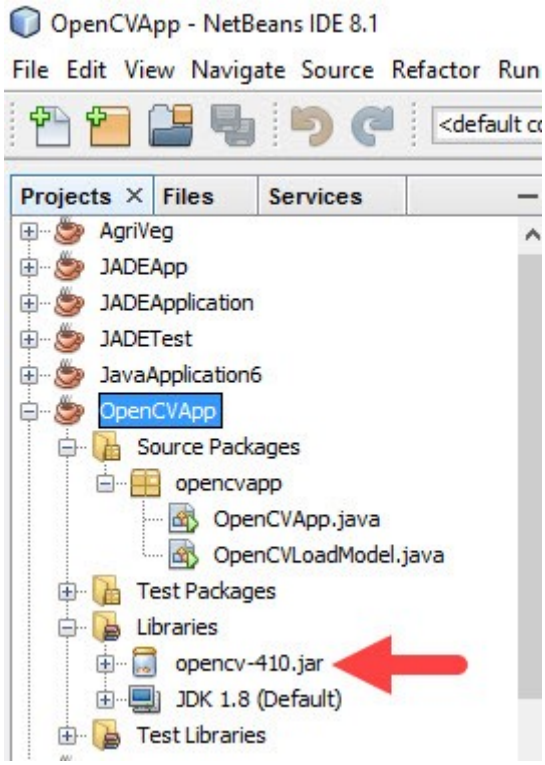
On the right of this window, click the **Add JAR/Folder** button which opens a new window for selecting the JAR file. Navigate to the location of the **opencv-410.jar** file, select it, and then click **Open.**



By clicking on the **Open** button, the selected JAR file will be listed. Click on **OK** to finish importing the JAR file.



Going to the project structure and expanding the Libraries directory, you can find the selected JAR file listed as a library. This indicates a successful importation of the JAR file.



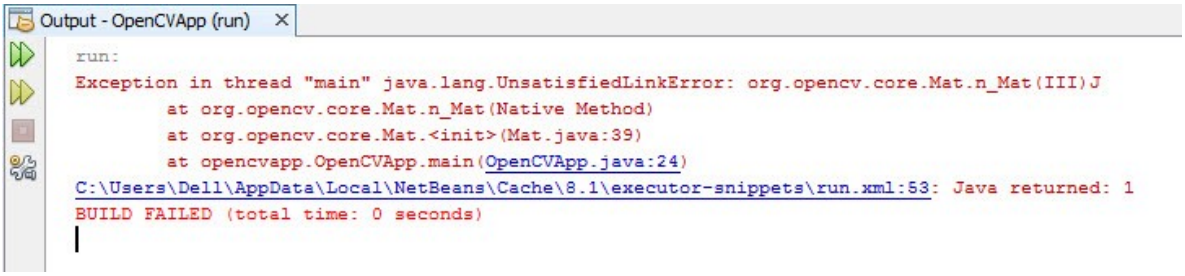
Based on the auto completion feature of NetBeans, you can find that it now detects OpenCV, as shown below.

After that, let's check to make sure OpenCV is working when used. We do this by creating an empty Mat, according to the code below.

```
package opencvapp;
import static org.opencv.core.CvType.CV_32F;
import org.opencv.core.Mat;
public class OpenCVApp {
public static void main(String[] args) {
    Mat test = new Mat(1, 1, CV_32F);
}
}
```

Solving the UnsatisfiedLinkError

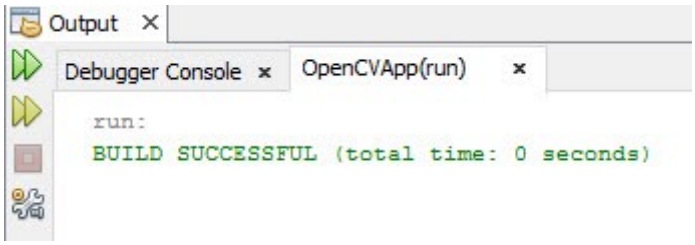
After running the project, an UnsatisfiedLinkError exception is thrown. It indicates that there's an error in linking OpenCV to the project. The problem arises because the OpenCV DLL file is not detected automatically, and thus it isn't loaded. This happens because the DLL path used by default by OpenCV is incorrect.



To fix this issue, we can load the DLL file manually by specifying the correct path within the `System.load()` method. It's very important to use the correct DLL based on your OS. If you're using a 64-bit OS, then use x64 within the path. For 32-bit, use x86 (I used x64 because I am using 64-bit OS).

```
package opencvapp;
import org.opencv.core.Core;
import static org.opencv.core.CvType.CV_32F;
import org.opencv.core.Mat;
public class OpenCVApp {
    public static void main(String[] args) {
        System.load("D:\\Opencv-4.1.0-vc14_vc15\\opencv\\build\\java\\x64\\" +
Core.NATIVE_LIBRARY_NAME + ".dll");
        Mat test = new Mat(1, 1, CV_32F);
    }
}
```

After running the class, it will run successfully as indicated in the next figure.



As of now, OpenCV is loaded correctly within NetBeans. Now that everything's loaded correctly, we'll start preparing training data for building the XOR gate using an ANN.

Preparing the Training Data

The ANN is trained to mimic the operation of the 2-input XOR gate. The gate returns 0 when both inputs are identical (either 0 or 1) and returns 1 when they are different (one input is 1 and the other input is 0). Because the output is either 0 or 1, it's a binary classification task. The truth table of the gate is given in the next table, which maps the inputs to their class label (output).

Input 1	Input 2	Out
0	0	0
0	1	1
1	0	1
1	1	0

In order to feed the training data (inputs and outputs) to the ANN build using OpenCV, the data must be stored into an OpenCV **Mat**. We can simply create a regular Java array for holding the data and then convert it into a Mat. The code for building the Mat for the training input data is shown below.

A 2D Java array named `XORTrainArray` is created for holding the 2 inputs. The array has 4 rows, one for each data sample. Each row has 2 columns, one for each input to the XOR gate.

After creating the Java array, next is to create an OpenCV **Mat** in which the array data are copied to it. The Mat is created by instantiating the Mat class. The constructor of this class accepts 3 integer arguments:

1. `int rows`: Number of rows
2. `int cols`: Number of columns
3. `int type`: Data type

Because there are 4 samples, the rows argument is assigned 4. The cols argument is assigned 2 because each sample has 2 inputs. Finally, the type argument is assigned `CV_32F`, which refers to a floating-point number stored into 32 bits (4 bytes). The OpenCV data types are stored into the `CvType` class available in the `org.opencv.core` package. By doing that, an empty Mat is created. The next step is to copy the Java array data into it.

```
package opencvapp;
import org.opencv.core.Core;
import static org.opencv.core.CvType.CV_32F;
import org.opencv.core.Mat;
public class OpenCVApp {
    public static void main(String[] args) {
        System.load("D:\\Opencv-4.1.0-vc14_vc15\\opencv\\build\\java\\x64\\" +
Core.NATIVE_LIBRARY_NAME + ".dll");
        double[][] XORTrainArray = {
            {0.0, 0.0},
            {0.0, 1.0},
            {1.0, 0.0},
            {1.0, 1.0}
        };
        Mat XORTrain = new Mat(4, 2, CV_32F);
        XORTrain.put(0, 0, XORTrainArray[0]);
        XORTrain.put(1, 0, XORTrainArray[1]);
```

```
        XORTrain.put(2, 0, XORTrainArray[2]);
        XORTrain.put(3, 0, XORTrainArray[3]);
        System.out.println("Train Inputs : \n" + XORTrain.dump());
    }
}
```

There's a method in the Mat class called `put()` that inserts data into the Mat given the row and column indices. It accepts 3 arguments:

1. `int row`: Start row index
2. `int col`: Start column index
3. `double… data`: Data to be inserted

For the first sample in the data array, the start row index will be 0 and the start column index will be also 0. The first sample in the array is fetched using `XORTrainArray[0]` which returns `[0.0, 0.0]`. This sample will be inserted starting from the first row and first column in the Mat. Because there's only 1 row in this sample, only the first row in the Mat will be occupied. But 2 columns will be used from the first row because the sample has 2 values.

For the second sample, the start column index will still be 0, but the row index will be 1. The second sample from the Java array will be returned by using the index 1 for `XORTrainArray`. The process continues until filling the Mat with the 4 samples. The last line prints the Mat as shown in the next figure.

```
run:
Train Inputs :
[0, 0;
 0, 1;
 1, 0;
 1, 1]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Similar to creating the training inputs and converting them into an OpenCV Mat, the training outputs will be also inserted into a Mat according to the code below. The Java array that holds the outputs is named `XORTrainOutArray`. The array contents are inserted into a Mat named `XORTrainOut`. It has 4 rows as the `XORTrain` Mat but only 1 column because there is only 1 class label (output) for each sample.

```
package opencvapp;
import org.opencv.core.Core;
import static org.opencv.core.CvType.CV_32F;
import org.opencv.core.Mat;

public class OpenCVApp {
    public static void main(String[] args) {
        System.load("D:\\Opencv-4.1.0-vc14_vc15\\opencv\\build\\java\\x64\\" +
Core.NATIVE_LIBRARY_NAME + ".dll");
        double[][] XORTrainArray = {
            {0.0, 0.0},
            {0.0, 1.0},
            {1.0, 0.0},
            {1.0, 1.0}
        };
        Mat XORTrain = new Mat(4, 2, CV_32F);
        XORTrain.put(0, 0, XORTrainArray[0]);
        XORTrain.put(1, 0, XORTrainArray[1]);
        XORTrain.put(2, 0, XORTrainArray[2]);
```

```
        XORTrain.put(3, 0, XORTrainArray[3]);
        System.out.println("Train Inputs : \n" + XORTrain.dump());

        double[][] XORTrainOutArray = {
            {0.0},
            {1.0},
            {1.0},
            {0.0}
        };
        Mat XORTrainOut = new Mat(4, 1, CV_32F);
        XORTrainOut.put(0, 0, XORTrainOutArray[0]);
        XORTrainOut.put(1, 0, XORTrainOutArray[1]);
        XORTrainOut.put(2, 0, XORTrainOutArray[2]);
        XORTrainOut.put(3, 0, XORTrainOutArray[3]);
        System.out.println("Train Labels : \n" + XORTrainOut.dump());
    }
}
```

The printed data (both inputs and labels) is given below. By doing this, the training data (inputs and outputs) are prepared and ready to be fed to the ANN. Next is to build and train the ANN.

```
run:
Train Inputs :
[0, 0;
 0, 1;
 1, 0;
 1, 1]
Train Labels :
[0;
 1;
 1;
 0]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Building the ANN (Architecture and Parameters)

From its name, Open Computer Vision, OpenCV is known to focus on building computer vision algorithms that run in real-time. Now, OpenCV is more than that. OpenCV now supports building and training machine learning (ML) algorithms using the **org.opencv.ml** package.

In terms of ANNs, the OpenCV library has a class named `ANN_MLP` inside the `org.opencv.ml` package for building ANNs. `ANN_MLP` stands for **Artificial Neural Network — Multi-Layer Perceptrons**.

The first step towards building an ANN using OpenCV is to create an empty ANN by calling the static `create()` method inside the `ANN_MLP` class. This returns an instance of this class that represents the ANN.

After creating this instance, the regular work for any ANN is to be done. This includes specifying the ANN architecture, selecting the activation function, training the network, and more. The code below extends the previous class to create an ANN and specify the network architecture. The instance of the `ANN_MLP` is returned into the `ANN` variable.

```
package opencvapp;
import org.opencv.core.Core;
import static org.opencv.core.CvType.CV_32F;
import static org.opencv.core.CvType.CV_8U;
import org.opencv.core.Mat;
```

```
import org.opencv.ml.ANN_MLP;

public class OpenCVApp {

    public static void main(String[] args) {
        System.load("D:\\Opencv-4.1.0-vc14_vc15\\opencv\\build\\java\\x64\\" +
Core.NATIVE_LIBRARY_NAME + ".dll");
        double[][] XORTrainArray = {
            {0.0, 0.0},
            {0.0, 1.0},
            {1.0, 0.0},
            {1.0, 1.0}
        };
        Mat XORTrain = new Mat(4, 2, CV_32F);
        XORTrain.put(0, 0, XORTrainArray[0]);
        XORTrain.put(1, 0, XORTrainArray[1]);
        XORTrain.put(2, 0, XORTrainArray[2]);
        XORTrain.put(3, 0, XORTrainArray[3]);
        System.out.println("Train Inputs : \n" + XORTrain.dump());

        double[][] XORTrainOutArray = {
            {0.0},
            {1.0},
            {1.0},
            {0.0}
        };
        Mat XORTrainOut = new Mat(4, 1, CV_32F);
        XORTrainOut.put(0, 0, XORTrainOutArray[0]);
        XORTrainOut.put(1, 0, XORTrainOutArray[1]);
        XORTrainOut.put(2, 0, XORTrainOutArray[2]);
        XORTrainOut.put(3, 0, XORTrainOutArray[3]);
        System.out.println("Train Labels : \n" + XORTrainOut.dump());

        ANN_MLP ANN = ANN_MLP.create();

        Mat layerSizes = new Mat(3, 1, CV_8U);
        layerSizes.put(0, 0, 2);
        layerSizes.put(1, 0, 2);
        layerSizes.put(2, 0, 1);
        ANN.setLayerSizes(layerSizes);
        System.out.println("Layers Sizes : \n" + layerSizes.dump());
    }
}
```

In order to specify the numbers and sizes of the network layers, the `setLayerSizes()` method is used. It accepts a vector representing the number of neurons within each layer. The vector length represents the number of layers, given that the input and output layers are counted. For example, if the vector length is 5, then the network has 1 input layer, 1 output layer, and 3 hidden layers.

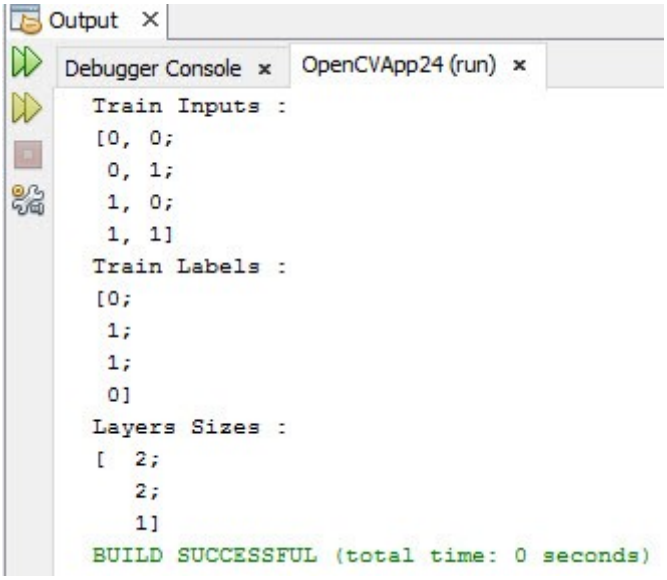
The vector is created according to the 1D Mat named `layerSizes`. It has 3 rows and just 1 column. Its data type is set to `CV_8U`, which corresponds to an unsigned 8-bit integer. Remember that all OpenCV data types are available in the `org.opencv.core.CvType` class.

The vector length is 3, and thus the network has a single hidden layer. The reason is that the optimal number of hidden layers for building the XOR gate is 1. Do you want to know why? Read my DataCamp tutorial titled **Deduce the Number of Layers and Neurons for ANN**.

The values inside the vector refer to the number of neurons for each layer. The first value at index 0 refers to the number of input neurons, the last value refers to the number of neurons in the output layers, and the in-between values refer to the number of neurons in the hidden layers.

Using the `put()` method, the values are inserted into the Mat. A value of 2 is inserted at the first location of the Mat (index 0) to reflect that 2 inputs are available in the input layer. At index 1, a value of 2 is inserted, which refers to the number of hidden neurons in the hidden layer. Note that 2 neurons in the hidden layer is the optimal choice for building an XOR gate. Finally, just one neuron is specified in the output layer because there is only 1 class label for each sample.

After running the above class, the `layerSizes` Mat is printed, as given below.



Up to this point, just the number of layers and their sizes are specified; but this isn't everything. Other parameters need to be specified. The code below specifies that the activation function type is `sigmoid` using the `setActivationFunction()` method. It also specifies that the training method is set to back propagation according to the `setTrainMethod()` method.

```
package opencvapp;
import org.opencv.core.Core;
import static org.opencv.core.CvType.CV_32F;
import static org.opencv.core.CvType.CV_8U;
import org.opencv.core.Mat;
import org.opencv.core.TermCriteria;
import org.opencv.ml.ANN_MLP;

public class OpenCVApp {

    public static void main(String[] args) {
        System.load("D:\\Opencv-4.1.0-vc14_vc15\\opencv\\build\\java\\x64\\" +
Core.NATIVE_LIBRARY_NAME + ".dll");
        double[][] XORTrainArray = {
            {0.0, 0.0},
            {0.0, 1.0},
            {1.0, 0.0},
            {1.0, 1.0}
        };
        Mat XORTrain = new Mat(4, 2, CV_32F);
        XORTrain.put(0, 0, XORTrainArray[0]);
        XORTrain.put(1, 0, XORTrainArray[1]);
```

```
        XORTrain.put(2, 0, XORTrainArray[2]);
        XORTrain.put(3, 0, XORTrainArray[3]);
        System.out.println("Train Inputs : \n" + XORTrain.dump());

        double[][] XORTrainOutArray = {
            {0.0},
            {1.0},
            {1.0},
            {0.0}
        };
        Mat XORTrainOut = new Mat(4, 1, CV_32F);
        XORTrainOut.put(0, 0, XORTrainOutArray[0]);
        XORTrainOut.put(1, 0, XORTrainOutArray[1]);
        XORTrainOut.put(2, 0, XORTrainOutArray[2]);
        XORTrainOut.put(3, 0, XORTrainOutArray[3]);
        System.out.println("Train Labels : \n" + XORTrainOut.dump());

        ANN_MLP ANN = ANN_MLP.create();

        Mat layerSizes = new Mat(3, 1, CV_8U);
        layerSizes.put(0, 0, 2);
        layerSizes.put(1, 0, 2);
        layerSizes.put(2, 0, 1);
        ANN.setLayerSizes(layerSizes);
        System.out.println("Layers Sizes : \n" + layerSizes.dump());

        ANN.setActivationFunction(ANN_MLP.SIGMOID_SYM);
        ANN.setTrainMethod(ANN_MLP.BACKPROP);

        TermCriteria criteria = new TermCriteria(TermCriteria.EPS +
TermCriteria.COUNT, 10000, 0.00000001);
        ANN.setTermCriteria(criteria);
    }
}
```

An ANN is an iterative algorithm in which the algorithm iterates through the data until reaching a good level of accuracy. To prevent such algorithms from going into infinite iterations, some other parameters need to be specified to determine when the algorithm terminates. OpenCV supports a class called `TermCriteria` which defines these parameters. The constructor of this class accepts 3 arguments:

1. `int type`: Termination type. Either stop when the error reaches a given limit, or when the number of iterations exceeds a maximum limit.
2. `int maxCount`: Maximum number of iterations.
3. `double epsilon`: The maximum difference between the desired and expected outputs. The lower the difference the more accurate the predictions.

The type could be one of these 3 values:

1. `TermCriteria.EPS`: Stop training when the desired error is reached.
2. `TermCriteria.COUNT`: Stop training when the maximum number of iterations is reached.
3. `TermCriteria.EPS + TermCriteria.COUNT`: Stop training when either the desired error is reached or when the maximum number of iterations is reached.

If the type is set to `TermCriteria.COUNT`, then the algorithm stops training when the number of iterations exceed the number assigned to the `maxCount` argument.

If the type is set to `TermCriteria.EPS`, then the algorithm stops training when the error (difference between the desired and predicted outputs) exceeds a specified value. This value should always be small. For example, 0.0001 or even smaller. The value used is 0.00000001.

If the termination type is set to `TermCriteria.EPS + TermCriteria.COUNT`, then the algorithm trains until the desired error is reached. If this error is not reached, then the algorithm stops after the specified number of iterations.

After specifying all parameters necessary for training the ANN, next we'll need to start training. Note that there are other parameters that might be necessary for you.

Training the ANN

In order to train the ANN, the `train()` method is called according to the modified class listed below. This method has different signatures, but the one used includes the following 3 arguments:

1. `InputArray samples`: Input training data
2. `int layout`: Sample layout.
3. `InputArray responses`: Output training data (labels)

The first argument is set to the training inputs Mat created previously inside the `XORTrain` variable. The last argument is set to the training outputs Mat stored into the `XORTrainOut` variable.

The second argument could be set to either `COL_SAMPLE` or `ROW_SAMPLE` based on how the sample is represented. If each sample is represented as a row, then the layout is set to `ROW_SAMPLE`. If each sample is represented as a column, then the layout is set to `COL_SAMPLE`. Throughout our discussion, each sample is represented as a row, and thus the used layout is `ROW_SAMPLE`.

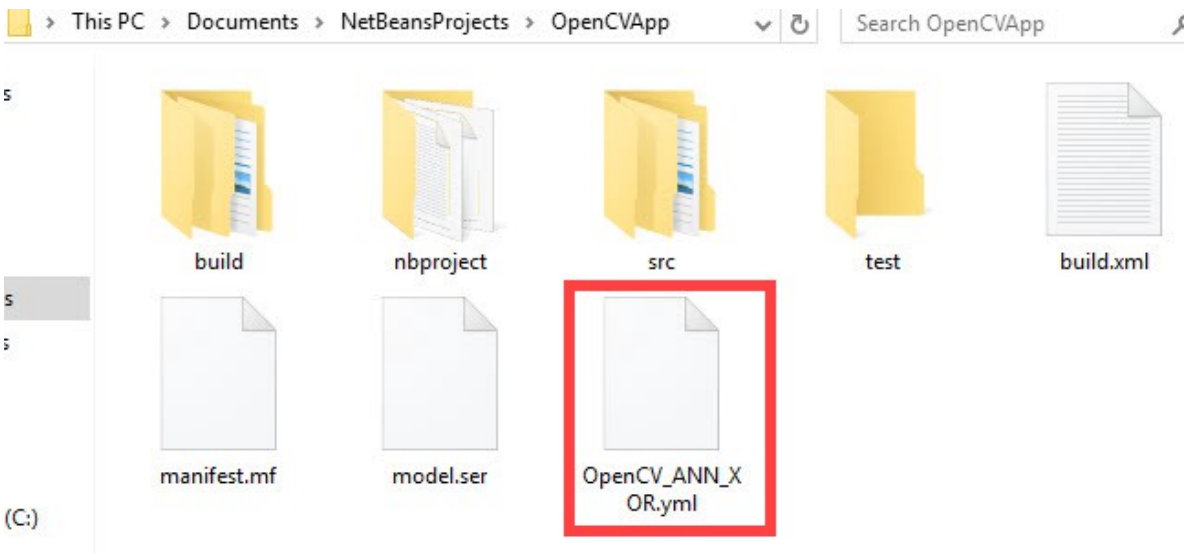
```
package opencvapp;
import org.opencv.core.Core;
import static org.opencv.core.CvType.CV_32F;
import static org.opencv.core.CvType.CV_8U;
import org.opencv.core.Mat;
import org.opencv.core.TermCriteria;
import org.opencv.ml.ANN_MLP;
import org.opencv.ml.Ml;

public class OpenCVApp {
    public static void main(String[] args) {
        System.load("D:\\Opencv-4.1.0-vc14_vc15\\opencv\\build\\java\\x64\\" +
Core.NATIVE_LIBRARY_NAME + ".dll");
        double[][] XORTrainArray = {
            {0.0, 0.0},
            {0.0, 1.0},
            {1.0, 0.0},
            {1.0, 1.0}
        };
        Mat XORTrain = new Mat(4, 2, CV_32F);
        XORTrain.put(0, 0, XORTrainArray[0]);
        XORTrain.put(1, 0, XORTrainArray[1]);
        XORTrain.put(2, 0, XORTrainArray[2]);
        XORTrain.put(3, 0, XORTrainArray[3]);
        System.out.println("Train Inputs : \n" + XORTrain.dump());
```

```
        double[][] XORTrainOutArray = {
            {0.0},
            {1.0},
            {1.0},
            {0.0}
        };
        Mat XORTrainOut = new Mat(4, 1, CV_32F);
        XORTrainOut.put(0, 0, XORTrainOutArray[0]);
        XORTrainOut.put(1, 0, XORTrainOutArray[1]);
        XORTrainOut.put(2, 0, XORTrainOutArray[2]);
        XORTrainOut.put(3, 0, XORTrainOutArray[3]);
        System.out.println("Train Labels : \n" + XORTrainOut.dump());

        ANN_MLP ANN = ANN_MLP.create();

        Mat layerSizes = new Mat(3, 1, CV_8U);
        layerSizes.put(0, 0, 2);
        layerSizes.put(1, 0, 2);
        layerSizes.put(2, 0, 1);
        ANN.setLayerSizes(layerSizes);
        System.out.println("Layers Sizes : \n" + layerSizes.dump());

        ANN.setActivationFunction(ANN_MLP.SIGMOID_SYM);
        ANN.setTrainMethod(ANN_MLP.BACKPROP);

        TermCriteria criteria = new TermCriteria(TermCriteria.EPS +
TermCriteria.COUNT, 10000, 0.00000001);
        ANN.setTermCriteria(criteria);

        ANN.train(XORTrain, Ml.ROW_SAMPLE, XORTrainOut);
        System.out.println("Model is trained? " + ANN.isTrained());

        Mat input_weights = ANN.getWeights(0);
        Mat hidden_weights = ANN.getWeights(1);
        Mat output_weights = ANN.getWeights(2);
        System.out.println("Input Layer Weights : \n" + input_weights.dump());
        System.out.println("Hidden Layer Weights : \n" + hidden_weights.dump());
        System.out.println("Output Layer Weights : \n" + output_weights.dump());

        try{
            ANN.save("OpenCV_ANN_XOR.yml");
            System.out.println("Model Saved Successfully.");
        } catch(Exception ex) {
            System.err.println("Error Saving Model.");
        }
    }
}
```

After the network is trained, the final weights can be fetched using the `getWeights()` method. It accepts the layer index and then returns its weights as a Mat. The weights of all 3 layers are fetched and printed, as shown below.

```
Input Layer Weights :
[2, -1, 2, -1]
Hidden Layer Weights :
[-2.613498354515718, -2.453098351475378;
 -2.577973014298946, -2.421471614473205;
 2.060408326223987, -1.766758326017668]
Output Layer Weights :
[2.355776228449655;
 -2.37040398727325;
 -2.690446661074365]
```

After training completes, the trained ANN model is saved in a YML file named `OpenCV_ANN_XOR.yml`. This enables us to use the trained model later for making predictions. The file is saved under the root of the NetBeans project directory, as shown below.



Loading and Testing the Trained ANN in NetBeans

After saving the trained model successfully, next we load it and make predictions according to testing samples. If you didn't create the `OpenCVLoadModel` class earlier, create it now. The contents of this class are listed below.

The class prepares the data exactly the same as done in the `OpenCVApp` class. The testing samples are the same as the training samples because the XOR gate has just 4 samples, and it's impossible to split its data into training and testing.

This class loads the model using the `load()` method. Predictions take place using the `predict()` method, which accepts 3 arguments:

1. `Mat samples`: Inputs of testing samples.
2. `Mat results`: Mat to save the prediction results.
3. `int flags`: Optional flags. Set to 0 if not required.

Note that the `results` Mat returned by this method represents not the predicted class label but a score. For example, it returns a score of 0.01. How to deduce the class label from this score? If this score is less than 0.5, then it refers to class 0. Otherwise, it refers to class 1. This is why there's a threshold applied to the values inside the `results` Mat.

A for loop iterates through the samples, feeds each sample to the `predict()` method, returns the result, thresholds it to return the predicted class label, compares it with the desired class label, and increases a counter named `num_correct_predictions` for each correct prediction.

Finally, the class prints the accuracy by dividing the value within this variable by the number of samples. Because each sample is represented as a row in the `XORTrain` Mat, the number of samples is equal to the number of rows within it. The number of rows is returned using the `rows()` method.

```
package opencvapp;
import org.opencv.core.Core;
import static org.opencv.core.CvType.CV_32F;
import org.opencv.core.Mat;
import org.opencv.ml.ANN_MLP;

public class OpenCVLoadModel {
    public static void main(String[] args) {
        System.load("D:\\FCI\\Programming\\OpenCV\\OpenCV 4.1.0\\opencv-4.1.0-
vc14_vc15\\opencv\\build\\java\\x64\\" + Core.NATIVE_LIBRARY_NAME + ".dll");

        double[][] XORTrainArray = {
            {0.0, 0.0},
            {0.0, 1.0},
            {1.0, 0.0},
            {1.0, 1.0}
        };
        Mat XORTrain = new Mat(4, 2, CV_32F);
        XORTrain.put(0, 0, XORTrainArray[0]);
        XORTrain.put(1, 0, XORTrainArray[1]);
        XORTrain.put(2, 0, XORTrainArray[2]);
        XORTrain.put(3, 0, XORTrainArray[3]);
        System.out.println("Train Inputs : \n" + XORTrain.dump());

        double[][] XORTrainOutArray = {
            {0.0},
            {1.0},
            {1.0},
            {0.0}
        };
        Mat XORTrainOut = new Mat(4, 1, CV_32F);
        XORTrainOut.put(0, 0, XORTrainOutArray[0]);
        XORTrainOut.put(1, 0, XORTrainOutArray[1]);
        XORTrainOut.put(2, 0, XORTrainOutArray[2]);
        XORTrainOut.put(3, 0, XORTrainOutArray[3]);
        System.out.println("Train Labels : \n" + XORTrainOut.dump());

        ANN_MLP ANN =
ANN_MLP.load("C:\\Users\\Dell\\Documents\\NetBeansProjects\\OpenCVApp\\OpenCV_AN
N_XOR.yml");

        double num_correct_predictions = 0;
        for (int i = 0; i < XORTrain.rows(); i++) {
            Mat sample = XORTrain.row(i);
            double correct_label = XORTrainOut.get(i, 0)[0];

            Mat results = new Mat();
            ANN.predict(sample, results, 0);

            double response = results.get(0, 0)[0];
            double predicted_label;
```

```
            if (response >= 0.5) {
                predicted_label = 1.0;
            } else {
                predicted_label = 0.0;
            }

            System.out.println("Input Sample : " + sample.dump() + ", Predicted
Score : " + response + ", Predicted Label : " + predicted_label + ", Correct
Label : " + correct_label);

            if (predicted_label == correct_label) {
                num_correct_predictions += 1;
            }
        }
        double accuracy = (num_correct_predictions / XORTrain.rows()) * 100;
        System.out.println("Accuracy : " + accuracy);
    }
}
```

The next figure shows that predicted scores of all 4 samples, in addition to the accuracy of the trained model, which is 100%. Note how the predicted scores for samples with 0 as the correct class label are very close to 0. Also the predicted scores for samples with 1 as the correct class label are very close to 1. The reason for reaching such results is selecting a small value for the `epsilon` argument in the constructor of the `TermCriteria` class, which is 0.00000001.

```
Input Sample : [0, 0], Predicted Score : 3.1647132709622383E-4, Predicted Label : 0.0, Correct Label : 0.0
Input Sample : [0, 1], Predicted Score : 0.9998542070388794, Predicted Label : 1.0, Correct Label : 1.0
Input Sample : [1, 0], Predicted Score : 0.9998565316200256, Predicted Label : 1.0, Correct Label : 1.0
Input Sample : [1, 1], Predicted Score : 3.0846064328216016E-4, Predicted Label : 0.0, Correct Label : 0.0
Accuracy : 100.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

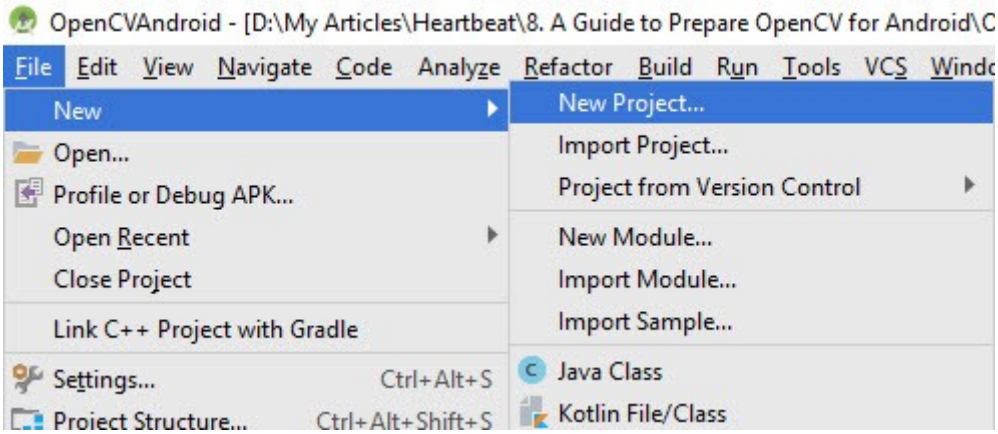
After making sure that the model is loaded successfully and it predicts the samples with high accuracy, we can deploy it to Android. So let's create an Android Studio project that loads the trained model and then execute the same code used in the `OpenCVLoadModel` class.

Building an Android Studio Project

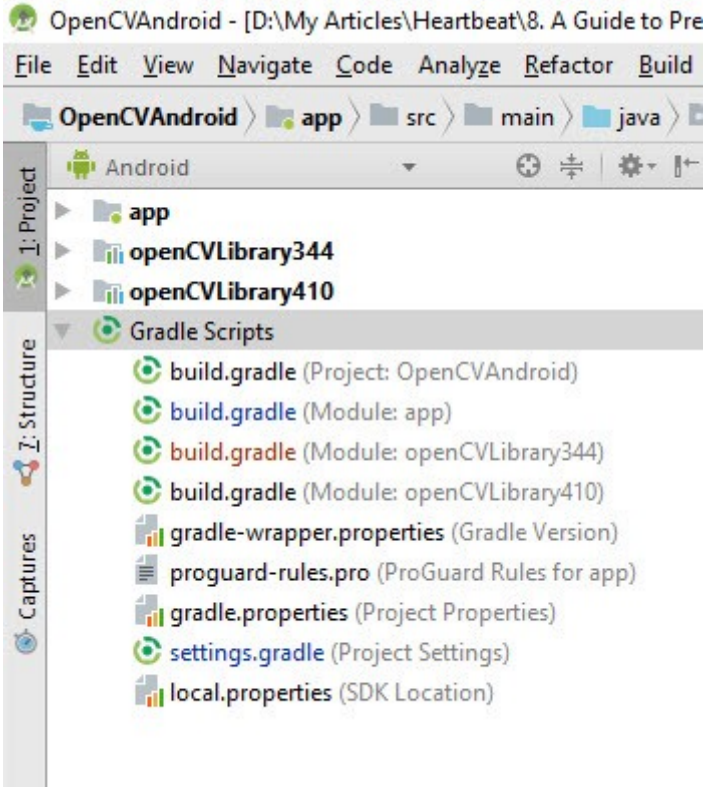
Is is expected that the reader already has an Android Studio project linked with OpenCV. If you do not yet have a project prepared to use OpenCV, then you can download the one available on GitHub at this link https://github.com/ahmedfgad/openCVAndroid.

For details about linking OpenCV to Android Studio, you can read the the post in section 1.1 titled **A Guide to Prepare OpenCV for Android**.

For creating a new Android Studio project, click on **File,** then **New,** and select **New Project**. Follow the steps of project creation and chose to create an activity with an empty layout.



After the project is created, load OpenCV and you'll see the Android view of the project.



In this case, I loaded both OpenCV 3.4.4 and OpenCV 4.1.0. Which one to use? This is specified inside the **dependencies** section of the `build.gradle` file of the application. As give below, I used OpenCV 3.4.4. If `openCVLibrary344` is replaced by `openCVLibrary410`, then OpenCV 4.1.0 will be used.

```
dependencies {
...

    implementation project(':openCVLibrary344')

}
```

After creating the project and making sure OpenCV is imported correctly within it, next we need to load the trained model and make predictions.

Loading the Trained ANN in Android Studio and Making Predictions

The XML layout of the main activity is listed below. There is a `Button` view and an `EditText` view. The user should enters the path of the trained YML model inside the `EditText` view. When the button is clicked, the callback method named `predictANN()` is called where the path is returned and fed to the `load()` model.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    tools:context="com.example.dell.opencvandroid.MainActivity">

    <Button
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="Load ANN using OpenCV"
        android:onClick="predictANN"
        />
    <EditText
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="/storage/emulated/0/OpenCV_ANN_XOR.yml"
        android:id="@+id/modelPath"/>

</LinearLayout>
```

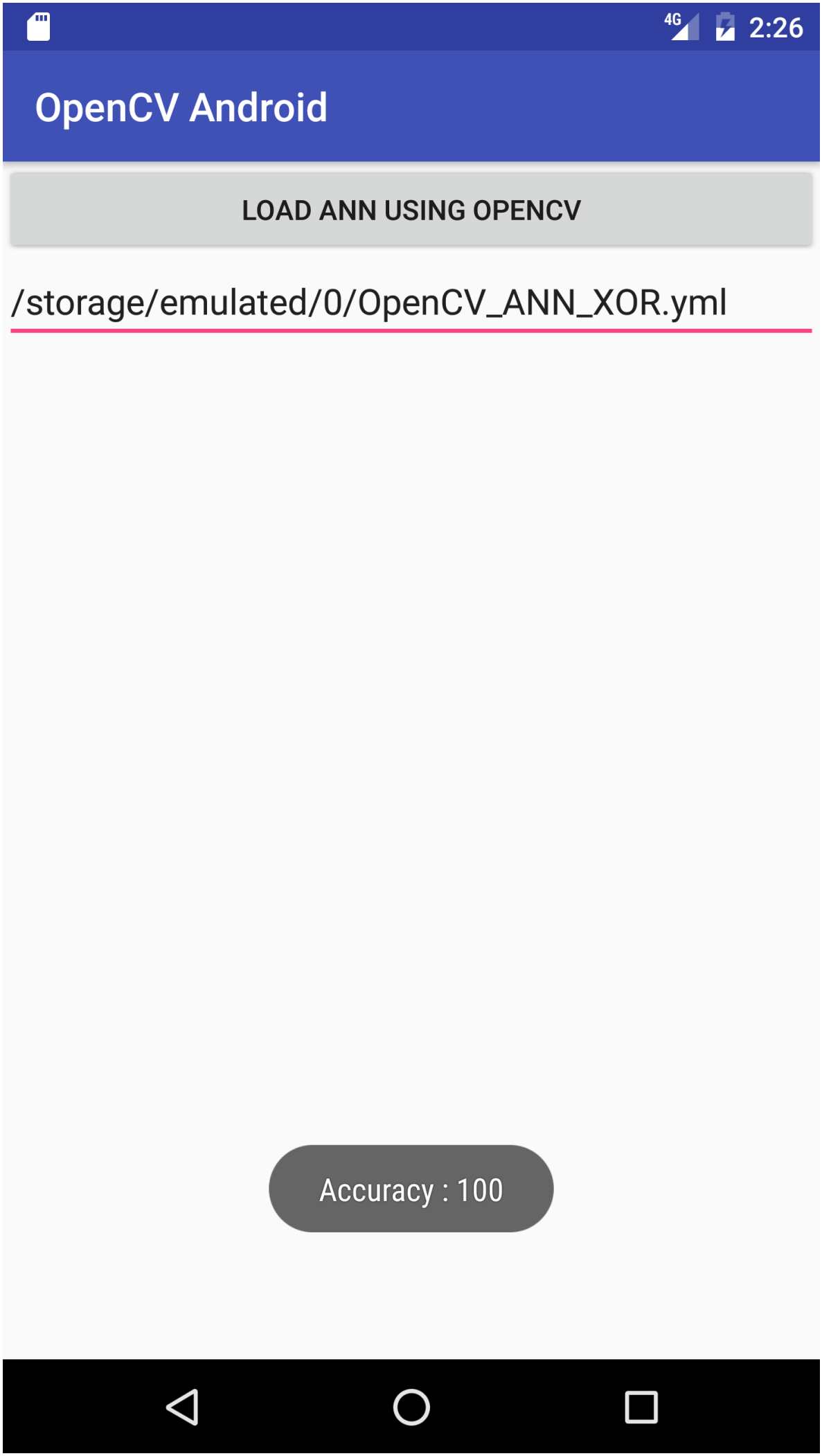
The implementation of the `predictANN()` method is found inside the `MainActivity` class listed below. Note that you have to upload the trained model (YML) file to your device and enter the path into the `EditText` view where it will be fetched and fed to the `load()` method.

```
package com.example.dell.opencvandroid;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;
import android.widget.EditText;
import android.widget.Toast;

import org.opencv.android.OpenCVLoader;
import org.opencv.core.Mat;
import org.opencv.ml.ANN_MLP;

import static org.opencv.core.CvType.CV_32F;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
//        OpenCVLoader.initAsync(OpenCVLoader.OPENCV_VERSION, this, null);
        OpenCVLoader.initDebug();
    }
```

```
    public void predictANN(View v){
        double[][] XORTrainArray = {
                {0.0, 0.0},
                {0.0, 1.0},
                {1.0, 0.0},
                {1.0, 1.0}
        };
        Mat XORTrain = new Mat(4, 2, CV_32F);
        XORTrain.put(0, 0, XORTrainArray[0]);
        XORTrain.put(1, 0, XORTrainArray[1]);
        XORTrain.put(2, 0, XORTrainArray[2]);
        XORTrain.put(3, 0, XORTrainArray[3]);
        System.out.println("Train Inputs : \n" + XORTrain.dump());

        double[][] XORTrainOutArray = {
                {0.0},
                {1.0},
                {1.0},
                {0.0}
        };
        Mat XORTrainOut = new Mat(4, 1, CV_32F);
        XORTrainOut.put(0, 0, XORTrainOutArray[0]);
        XORTrainOut.put(1, 0, XORTrainOutArray[1]);
        XORTrainOut.put(2, 0, XORTrainOutArray[2]);
        XORTrainOut.put(3, 0, XORTrainOutArray[3]);
        System.out.println("Train Labels : \n" + XORTrainOut.dump());

        EditText modelPath = findViewById(R.id.modelPath);
        ANN_MLP ANN = ANN_MLP.load(modelPath.getText().toString());

        double num_correct_predictions = 0;
        for (int i = 0; i < XORTrain.rows(); i++) {
            Mat sample = XORTrain.row(i);
            double correct_label = XORTrainOut.get(i, 0)[0];

            Mat results = new Mat();
            ANN.predict(sample, results, 0);

            double response = results.get(0, 0)[0];
            double predicted_label = 0.0;

            if (response >= 0.5) {
                predicted_label = 1.0;
            } else {
                predicted_label = 0.0;
            }

            System.out.println("Input Sample : " + sample.dump() + ", Predicted
Score : " + response + ", Predicted Label : " + predicted_label + ", Correct
Label : " + correct_label);

            if (predicted_label == correct_label) {
                num_correct_predictions += 1;
            }
        }

        double accuracy = (num_correct_predictions / XORTrain.rows()) * 100;
```

```
        Toast.makeText(getApplicationContext(), "Accuracy : " + accuracy,
Toast.LENGTH_LONG).show();

    }
}
```

After launching the app and clicking the button, the result is shown in the next figure. A Toast message prints the accuracy, which is 100.


2:26
OpenCV Android
LOAD ANN USING OPENCV
/storage/emulated/0/OpenCV_ANN_XOR.yml
Accuracy : 100

Conclusion

We discussed how to build ANN in Android devices using OpenCV. By doing this, there are much more apps to build. For example, capture images using Android camera and classify them using the ANN running in Android. In section 1.3, OpenCV will be used in Android for extracting features from images. The extracted features will be fed into a pre-trained ANN for classification.