## Tensorflow in Java mit BlueJ

Mit der folgenden Anleitung kann ein bereits fertig trainiertes neuronales Netz zur Bilderkennung aus dem offiziellen Tensorflow-Model-Pool in Java verwendet werden.

Unter https://tfhub.dev werden zahlreiche bereits trainierte neuronale Netze aus den Bereichen Text, Bilder, Video und Audio zur Verfügung gestellt, die kostenfrei heruntergeladen werden können. Im Folgenden beschränken wir uns jedoch auf den Bereich Bilder und nutzen ein von Google bereitgestelltes neuronales Netz in Java, um Bilder zu klassifizieren. Diese Anleitung funktioniert für viele der im Bereich „Image“ angebotenen Modelle, möglicherweise jedoch nicht für alle Modelle.

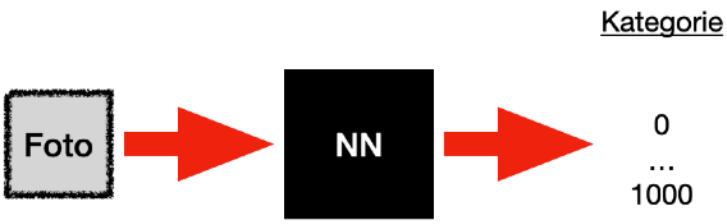
### 1. Model herunterladen

Ein gut funktionierendes und bekanntes Modell, um Bildklassifikation durchzuführen ist das neuronale Netz mit dem Namen MobileNet V3. Es befindet sich in der image-Sammlung von Google und kann unter folgendem Link heruntergeladen werden:

https://tfhub.dev/google/imagenet/mobilenet_v3_large_100_224/classification/5

Auf dieser Seite finden sich noch die Informationen, dass das neuronale Netz Bilder mit einer Größe von 224x224 Pixeln verarbeiten und diese in eine von 1000 Klassen klassifizieren kann. Eine Textdatei mit den 1000 Klassen (so genannte class labels) kann ebenso auf der Seite heruntergeladen werden und wird im Folgenden auch benötigt.

Das neuronale Netz übernimmt also folgende Aufgabe: Es liest ein 224x224 großes Foto und gibt am Ende ein Array mit 1000 Stellen aus. Dabei erhält diejenige Stelle im Array den höchsten Wert, die zu der passenden Klasse aus der Textdatei gehört und für die vermutet wird, dass das Foto diese Klasse enthält. Wird z.B. ein Foto einer Ananas klassifiziert, erhält bei korrekter Klassifikation die Stelle 954 im Array den höchsten Wert, da der 954. Eintrag in der Textdatei *„Pineapple“* ist. Die in dieser Anleitung bereitgestellte Klasse ermöglicht es übrigens, direkt den Namen der klassifizierten Klasse zurückzugeben. So kann z.B. ein Foto gelesen werden, auf dem eine Ananas abgebildet ist und es wird am Ende ein String-Objekt mit dem Wert "Ananas" zurückgegeben.



## 2. Vorbereitung

Es bietet sich an, einen Ordner auf der Festplatte einzurichten, in dem sämtliche Dateien vorhanden sind, und zwar: Der heruntergeladene und entpackte Ordner mit dem neuronalen Netz, optional die Textdatei mit den „class labels“ und ein oder mehrere Testbilder /-fotos.

Dieser Ordner sollte also folgende Dinge enthalten (Ordner und Unterordner in **fett**):

**MeinOrdner**
- **imagenet_mobilenet_v3_large_100_224_classification_5**
  - **assets**
  - saved_model.pb
  - **variables**
    - variables.data-00000-of-00001
    - variables.index
- ImageNetLabels.txt
- testbild.png *

Weiterhin muss (z.B. an anderer Stelle) der Ordner mit dem BlueJ-Projekt vorhanden sein. Dieser enthält einen Unterordner namens lib, der mit den notwendigen Bibliotheken gefüllt sein muss, was im Anschluss hieran erläutert wird. Dieser Ordner sollte insgesamt folgende Dinge enthalten (Ordner und Unterordner in **fett**, die Versionsnummern der Bibliotheken können ggf. abweichen):

**TF**
- Demo.class
- Demo.ctxt
- Demo.java
- **lib**
  - javacpp-1.5.7.jar
  - ndarray-0.3.3.jar
  - protobuf-java-4.0.0-rc-2.jar
  - tensorflow-core-api-0.4.0-macosx-x86_64.jar **
  - tensorflow-core-api-0.4.0.jar
  - tensorflow-framework-0.4.0.jar
- package.bluej
- README.TXT
- TFImageModel.class
- TFImageModel.ctxt
- TFImageModel.java

* ein 224x224 Pixel großes Bild einer beliebigen Kategorie

** je nach Betriebssystem anders

### 3. Die Tensorflow Bibliotheken herunterladen

Um Tensorflow Modelle in Java nutzen zu können, müssen die entsprechenden Bibliotheken heruntergeladen werden. Folgende sechs jar-Dateien müssen in den Unterordner lib im BlueJ-Projekt heruntergeladen werden (im Folgenden wird der Dateiname beispielhaft angezeigt, die Versionsnummer kann abweichen):

1. TensorFlow Core API Library: org.tensorflow -> tensorflow-core-api
   https://mvnrepository.com/artifact/org.tensorflow/tensorflow-core-api
   -> **tensorflow-core-api-0.4.0.jar**

2. bei 1. zusätzlich mit Klick auf "View All" folgende Seite öffnen
   https://repo1.maven.org/maven2/org/tensorflow/tensorflow-core-api/0.4.0/
   Dort die entsprechende Library für das Betriebssystem herunterladen, z.B.:
   -> **tensorflow-core-api-0.4.0-macosx-x86_64.jar**
   -> **tensorflow-core-api-0.4.0-windows-x86_64.jar**
   -> **tensorflow-core-api-0.4.0-linux-x86_64.jar**

3. TensorFlow Framework Library:  org.tensorflow -> tensorflow-framework
   https://mvnrepository.com/artifact/org.tensorflow/tensorflow-framework/0.4.0
   -> **tensorflow-framework-0.4.0.jar**

4. Protocol Buffers [Core]: com.google.protobuf -> protobuf-java
   https://mvnrepository.com/artifact/com.google.protobuf/protobuf-java
   -> **protobuf-java-4.0.0-rc-2.jar**

5. JavaCPP: org.bytedeco -> javacpp
   https://mvnrepository.com/artifact/org.bytedeco/javacpp
   -> **javacpp-1.5.7.jar**

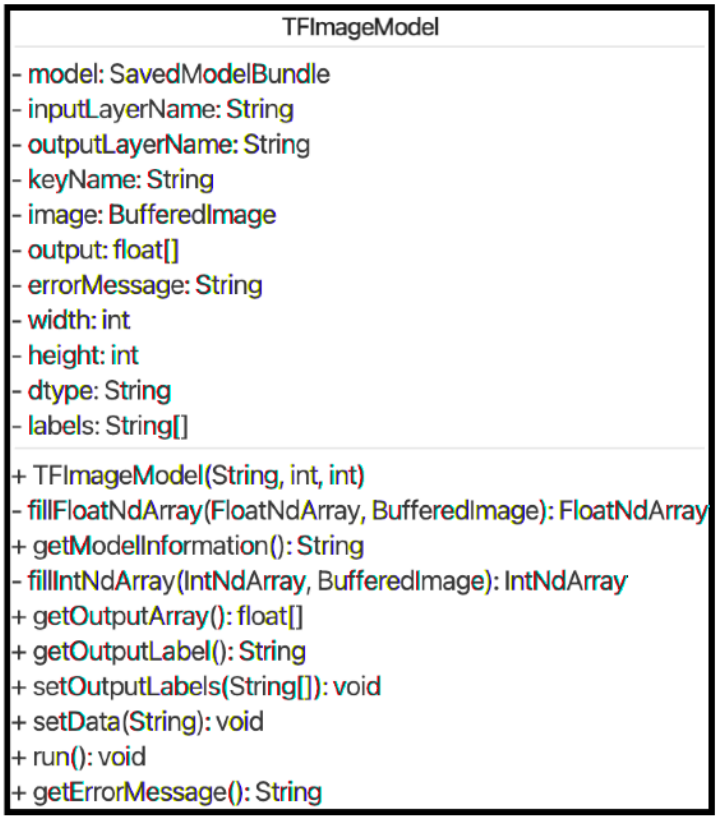
6. TensorFlow NdArray Library:  org.tensorflow -> ndarray
   https://mvnrepository.com/artifact/org.tensorflow/ndarray
   -> **ndarray-0.3.3.jar**

### 4. BlueJ einrichten

BlueJ muss vor der Verwendung zunächst eingerichtet werden. Die heruntergeladenen Bibliotheken müssen noch richtig verlinkt werden. Je nach Betriebssystem sind die Einstellungen ggf. an etwas anderer Stelle zu finden.

Das BlueJ-Projekt öffnen, dann im Menü auf BlueJ > Einstellungen (unter Linux & Windows: Werkzeuge > Einstellungen) klicken. Auf den Reiter Bibliotheken klicken. Dann auf Datei hinzufügen klicken. Nun die sechs jar-Dateien aus dem Ordner lib hinzufügen, BlueJ neustarten und das Projekt kompilieren. Es sollten nun keine Fehler mehr gemeldet werden.

### 5. Die Klasse TFImageModel nutzen




Eine Klassifikation läuft prinzipiell in folgenden Schritten ab:

1. **Model laden**
2. **Ein Bild ins Model laden**
3. **Die Klassifikation durchführen**
4. **Das Ergebnis abrufen**

Diese Schritte werden im Folgenden erläutert. Es existiert ebenso eine Demo-Klasse, die noch weiteren Code enthält, der zeigt, wie die Klassenlabels als Textdatei gelesen und ins Model geladen werden können.

**1. Model laden, indem der Pfad zum Model-Ordner übergeben wird, z.B.:**

```
a)    TFImageModel im = new TFImageModel("/Users/ich/Downloads/imagenet_mobile
            net_v3_large_100_224_classification_5", 224, 224);  //Mac & Linux

b)    TFImageModel im = new TFImageModel("C:\\Downloads\\imagenet_mobilenet_v3_
            large_100_224_classification_5", 224, 224);  //Windows
```

(optional: Die Datei mit den Klassenlabels laden, als String[] konvertieren und ins Modell laden)

```
      String[] labels =  ...
      im.setOutputLabels(labels)
```

**2. Ein Bild ins Model laden, z.B.:**

```
a)    im.setData("/Users/ich/Downloads/pineapple.png");

b)    im.setData("C:\\Downloads\\pineapple.png");
```

**3. Die Klassifizierung starten:**

```
      im.run();
```

**4. Das Ergebnis der Klassifikation abrufen, z.B.:**

i. wenn zuvor Klassenlabels in das Model geladen wurden:

```
      String ergebnis = im.getOutputLabel();
```

ii. wenn lediglich das Ausgabe-Array abgerufen werden soll:

```
      float[] erg = im.getOutputArray();
```

**Achtung: Gültige Pfadangaben in Java sind …**

- unter Windows z.B. "**C:\\Daten\\MeinOrdner\\myImage.png**"
- unter Mac OS / Linux z.B. "**/Users/myname/MeinOrdner/myImage.png**"

### 6. Das Demo-Programm ausführen

Die Demo-Klasse enthält eine main-Methode, die ausgeführt werden kann und eine Klassifikation durchführt, dazu muss der Quelltext nur geringfügig angepasst werden (z.B. die in rot gefärbten Pfadangaben), so dass die richtigen Dateien auch korrekt verlinkt sind. Gezeigt wird dies am obigen Ordnerstruktur-Beispiel für Mac OS & Linux:

```
public static void main(String[] args) {

    TFImageModel im = new TFImageModel("/…/MeinOrdner/imagenet_mobilenet_v3_lar
                                       ge_100_224_classification_5", 224, 224);

    String[] labels = null;
    try {
        List<String> result = Files.readAllLines(Paths.get("/…/MeinOrdner/
                                                            ImageNetLabels.txt"));
        labels = result.toArray(new String[result.size()]);
    }
    catch (java.io.IOException ioe) {
    }
    im.setOutputLabels(labels);

    im.setData("/…/MeinOrdner/testbild.png");
    im.run();

    System.out.println(im.getOutputLabel());

}
```

… bzw. als zweite Möglichkeit … :

```
public static void main(String[] args) {

    TFImageModel im2 = new TFImageModel("/…/MeinOrdner/imagenet_mobilenet_v3_lar
                                        ge_100_224_classification_5", 224, 224);

    im2.setData("/Users/ich/TF/data/pin.png");
    im2.run();

    float[] erg = im2.getOutputArray();
    if (erg != null) {
        for (int i=0; i<erg.length;i++) {
            System.out.println(i + "\t " + erg[i]);
        }
    }

}
```

## 7. Weitere Ideen

- Es ist möglich, ein fremdes oder eigenes in Python erstelltes neuronales Netz als SavedModel zu speichern, herunterzuladen und mit der bereitgestellten Klasse direkt in Java zu nutzen. Das gilt auch für Modelle, die in Jupyter-Notebooks auf colab.research.google.com erstellt wurden, wie z.B.:

  https://colab.research.google.com/github/tensorflow/docs/blob/master/site/en/tutorials/keras/classification.ipynb.

  Dies geht im Jupyter-Notebook einfach mit der Anweisung `model.save('my_model')`

- Es ist möglich, ein neuronales Netz unter https://teachablemachine.withgoogle.com direkt im Browser zu trainieren, zu exportieren und direkt mit der bereitgestellten Klasse in Java zu nutzen. Dazu beim Export den Reiter Tensorflow und danach den Punkt SavedModel auswählen. Man erhält einen Ordner mit der exakt gleichen Struktur wie auf https://tfhub.dev.

## 8. Übersicht der Methoden

**TFImageModel**(pPfad : String pHeight : int, pWidth : int)

Konstruktor, erwartet den Pfad zum Model und die Bildgröße, z.B.:

```
TFImageModel im = new TFImageModel("C:\\myModelFolder", 224, 224);
```

**setData**(pFilename : String) : void

Methode, um ein Bild zu laden, erwartet den Pfad zum Bild, z.B.:

```
im.setData("C:\\myImage.png");
```

**run**() : void

Führt die Klassifikation durch, z.B.:

```
im.run();
```

**setOutputLabels**(pLabelArray : String[]) : void

Lädt Klassenlabels, die bei der Ausgabe benutzt werden können, z.B.:

```
im.setOutputLabels(myStringArray);
```

**getOutputLabel**() : String

Gibt das Label der klassifizierten Klasse zurück, sofern Klassenlabels geladen wurden, z.B.:

```
String s = im.getOutputLabel();
```

**getOutputArray**() : float[]

Liefert das Klassifikationsergebnis als Array zurück. Die Länge des Arrays entspricht der Anzahl der Ausgabeneuronen, z.B.:

```
float[] erg = im.getOutputArray();
```

**getModelInformation**() : String

Liefert Informationen über das geladene Model als String zurück

**getErrorMessage**() : String

Liefert sämtliche Fehler, die bisher aufgetreten sind als String zurück