Department 07
Master Computer Science

Deep learning - Dog Breed Classification

Realization of an native Android app using deep learning algorithms

Alice Bollenmiller, Andreas Wilhelm
WS 17/18 IG
January 27, 2018

Contents

1. Introduction

In this chapter, a short introduction leads to the subject of Deep learning. Furthermore, the scope of this work and its points of referencee are described and localized.

1.1. Deep learning

In 2015, AlphaGo - a computer program developed by Google's DeepMind Group that was trained to play the strategic board game Go - was the first program to defeat the multiple European champion Fan Hui under tournament conditions. Back then, terms like Artifical Intelligence (AI), Machine Learning and Deep Learning were talked of, because those were the reason why a computer program was able to defeat a human being. One of the most frequent used techniques of AI is Machine Learning. It uses algorithms to parse data, process it and learn from it. The result is a prediction or determination as a conclusion of what was learnt from the dataset. Deep Learning - which is part of the Machine Learning techniques - sells its application particularly in the field of language and image processing.

In 1958, Frank Rosenblatt introduced the concept of the perceptron which is the fundamental idea of all Deep learning approaches. It consists of multiple artifical neurons which are coupled with weights and biases. In the case of a single-layer perceptron, the input nodes are fully connected to one or more output nodes. During the learning process the weights are adapted according to the learning progress. When such a structure is extended with layers, it becomes a multi-layer perceptron (MLP). This basic structure can be found in special neural network architectures e.g. Convolutional Neural Networks (CNN). CNNs which are frequently used for object detection and image/ audio recognition etc. consists of multiple neurons. When a neuron receives an input, it performs a dot production and adapts the weights. Because of their special structure, CNNs are able to detect local properties of an image. Basically, the network represents a differentiable score function which is applied on the raw image pixels and computes the class scores as an output.

1.2. Terms of Referencee

The problem of Deep learning architectures is their high performance requirements regarding computational power. Common frameworks for open source development in the field of deep learning which are discussed in section 2.1 introduced several models for the integration in mobile applications.

In order to overcome the above mentioned problem, optimizing methods will be combined with appropriate models which require less computing power and are suitable for mobile application integration. As a result of this work, a dog breed analyzer will be implemented. This mobile app will take a live camera stream as an input and determine the breed of the focused dog. The three highest probabilities of breeds will be shown by the app.

2. Methodological fundamentals

This chapter describes the most frequently used frameworks in deep learning for developing applications. Furthermore, common models for deep learning are introduced followed by suitable models for mobile integration. The chapter closes listing key requirements for an appropriate dataset which increase the quality of the training results.

2.1. Common Frameworks for Deep Learning Applications

The demands on neural networks increases with the complexity of problems to solve. Concurrently, there's an expanding offer of deep learning frameworks with a varity of features and tools. The most common used ones are represented in the following section.

2.1.1. Tensorflow

In 2015, the Google Brain Team introduced the most popular deep learning API Tensorflow which is an open-source library for numerical computation. Its current version 1.4.1 was released on December 8th, 2017. Tensorflow is primarily used for machine learning and deep neural network research. Based on the programming language Python, Tensorflow is capable of running on multiple CPUs and GPUs. Furthermore, C++ and R are supported by Tensorflow. Another feature is the possibility to generate models and export them as .pb file which holds the graph definition (GraphDef). The export is done by protocol buffers (protobuf) which includes tools for serializing and processing structured data. When loading a .pb file by protobuf, an graph object is created which holds a network of nodes. Each of those nodes represent an operation and the output is used as input for another operation. This concept enables an user to create self-built tensors. To simplify the usability, Tensorflow developed a high-level wrapper of the native API which is calles Tensorflow Slim. Futhermore, in order to run Tensorflow on performance critical devices like e.g. mobile devices there are two lightweight solutions of Tensorflow are available: Tensorflow Mobile and Tensorflow Lite. The latter one is an evolution of Tensorflow Mobile and still in developer mode. But both are predestinated for integration in mobile applications.

2.1.2. Keras

In order to simplify the utilization of Tensorflow the Python based interface Keras can be configured to work on top of Tensorflow. It allows building neural networks in a simple way and is part of Tensorflow.

2.1.3. Caffe

Another deep learning library is Caffe which was developed by Berkeley AI Research (BAIR). Based on C++ or Python, it focuses on modeling CNNs. An main advantage of Caffe is the offer of pretrained models available in the Caffe Model Zoo.

2.1.4. Torch and PyTorch

Besides Tensorflow and Caffe, Torch is another common deep learning framework. It was developed by Facebook, Twitter and Google. Based on C/C++, Torch supports CUDA for GPU processing. Like above mentioned frameworks, Torch facilitates the building of neural networks. The Python based version of Torch is available through PyTorch.

2.2. Common Models in Deep Learning Applications

State of the art models for image recognition in deep learning applications are in a CNNs architecture. The most important ones are explained in this section.
The first one was LeNet-5 of **?**. As exactly shown in Figure 1 it comprises seven layers. Thereof two times a convolutional layer with a 5x5 filter each followed by a sub-sampling pooling layer. Two fully connected layers complete the architecture of the multilayer perceptron with its softmax function applied.
In 2012 a CNN based network called AlexNet of **?** wons the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) for the first time (Figure 2). From that time CNN became ubiquitous.

Figure 1: Model architecture of LeNet-5 (**?**)

Figure 2: ImageNet Classification top-5 error (%) (**?**)

Through its immensely improved accuracy the network enabled a lot new possibilities. This CNN now consists of eight layers shown in Figure 3. The first convolutional layer has a large 11x11 filter. Furthermore one with 5x5 and another three with a 3x3 filter, so in total 5 convolutional layers. Three pooling layers and also three fully connected layers completes the architecture. For the first time a rectified linear unit (ReLU) was used and it could be spread across two GPUs.

Figure 3: Model architecture of AlexNet (**?**)

As seen in Figure 2 the next groundbreaking improvements were two architecures in 2014 which were also considered as very deep networks. VGG of **?** and GoogLeNet of **?** which is today better known under 'Inception'.
The first one comprises 16 or 19 layers in total (Figure 4. This network is characterized by its simplicity because it uses convolutional layers with a 3x3 filter which only stacked one above the other. Reducing the size is handled by pooling layers. At the end there are also three fully connected layers with the softmax function following.

Figure 4: Model architecture of VGG19 (**?**)

The other one has 22 layers (Figure 5) and was the final winner in 2014. GoogLeNet starts with two convolutional layers followed by stacked inception modules. Such a module consists of several parallel convolutional layers with different filter size and and also one pooling layer as shown in Figure 6. On the right side of this figure a convolutional layer with a 1x1 filter is used to reduce feature depth. On the lefts side all output of these modules are concatenate and forwarded as new input for the next layer. The network ends with only one fully connected layer.
Since 2015 it gets very deep with a new architecture named ResNet of **?**. On ILSVRC this network swept all competitiors in classification and detection. The networks itself is designed as a 152 layer model. The ResNet consists of many residual network blocks (Figure 7). Each of them has two convolutional layers with a 3x3 filter. Furthermore there is a shortcut connection which will be

Figure 5: Model architecture of Inception (**?**)

Figure 6: Inception modules (**?**)

used if the input and output dimension of this block are the same.

Figure 7: ResNet block (**?**)

Especially for performance critical applications like mobile devices MobileNet was developed by Google respectively **?**. The network structure consists of depthwise seperable convolutional parts but also one standard convolutional one at the beginning. The depthwise part is separated in a depthwise convolutional layer and a pointwise one which applies a 1x1 convolution. Each of this layers are followed by batchnormalisation and ReLu as shwon in Figure 8. In the depthwise convolution the splitting of the two convolutional layers in one for filtering and the other one for combining has the effect to reduce massively computation and model size which results in better performance on low performance devices.

Figure 8: Standard convolution and depthwise seperable convolution (**?**)

2.3. Key requirements for an appropriate dataset

Supervised learning tasks such as image recognition are based on operations where an output is taken as an input for the next node. Every raw pixel input is taken to compute an intermediate representation - a vector containing all learned information about the dataset. As a consequence, the training results are only as good as the dataset itself. For better accuracy its important to train a model on a variety of images for each object which should later be classified by the model. It's recommended to take images of an object which were taken at different times, with different devices and at different places. Otherwise, the model will concentrate on other things like for example the background instead of details about the object itself. Therefore, a huge dataset is required especially for non pre-trained models. Training a model from scratch will require a huge dataset, a lot computing power and time. Whereas pre-trained models only require a small dataset

of about hundreds of images. For that reason, a pre-trained model will be used in this work.

3. Concept

First, this chapter describes the selection of the appropriate framework. Futhermore, the structure of the model which was used for classification is explained based on its architecture. The chapter closes with the class diagram of the mobile application.

3.1. Frameworks

As a deep learning framework Tensorflow was used to retrain the model. This decision was mainly based on recommendations. Even companies like e.g. NVIDIA Corporation, Intel Corporation etc. use Tensorflow. It is one of the common frameworks for deep learning applications and also provides solutions for integration in mobile apps. Furthermore, Tensorflow provides a variety of tutorials for working with neural networks. Beside those advantages, there is a large community about Tensorflow talking about issues and solutions.

To run the tensor within a mobile application, the first approach was to use Tensorflow Lite which is still in development state. But many attempts resulted in corrupt models which caused the app to terminate. Because of this experiences Tensorflow Mobile was used to optimize the model for app integration.

3.2. Qualified models for mobile app integration

In Tensorflow Lite only InceptionV3 and MobileNet models are supported. With Tensorflow Mobile it's the same. There are different versions of MobileNet. They differ in the input image size and in the number of parameters which is also proportional to the size and needed computation power of the network.
While the Inception model was well known and established, the MobileNet is relatively new. InceptionV3 models gets an higher accuracy than MobileNets but MobileNets are more optimized on small size, low latency and low-power consumption which are important characteristics for mobile usage. (**?**)

In Figure **??** the MobileNet and Inception is compared with each other and also with other popular models already mentioned before in section 2.2. It shows the Top-1 accuracy and the Multiply-Accumulates (MACs) which measures the number of fused Multiplication and Addition operations. The latter numbers reflect the latency and power consumption of the network and so in result the efficiency. For well comparison reasons between MobileNet and Inception, a MobileNet model with similiar accuracy to the Inception one was picked, to be more precisely the MobileNet_1.0_-224. To check out the possibilities of MobileNet, another model, the MobileNet_0.50_244, was selected, too.

3.3. Application based Architecture

Alice This chapter describes the functions to a basic understanding of how the application works based on the application's architecture. It focuses on the important classes and methods of the mobile application.
Tensorflow provides a mobile application for demo purposes. Because of the complexity the application was adapted to our needs. The approach was to understand how the application was implemented regarding the functionality of the camera, image input for the network and background classification besides live camera stream.

Figure 9: Comparison of popular models (**?**)

The ClassifierActivity is set as the launcher activity of the application. It extends the class CameraActivity and loads the CameraFragment which controls the camera view. The CustomTextureView enables the possibility to capture frames from a camera stream and process it. Because of this function, the CustomTextureView is part of the class CameraFragment. If an image is captured, the method onImageAvailable of the class ClassifierActivity is invoked. The classification itself is done by the TensorflowImageClassifier which implements the interface Classifier from the Tensorflow API. The results are displayed by the class RecognitionScoreView which implements the interface ResultsView.

4. Realization

In general, this chapter describes the methodical procedure of solving the above mentioned problem in section 1.2. After describing the used dataset all required software and hardware components are explained in detail. Furthermore, the chapter leads through the installation steps of Tensorflow and the setup of Android Studio. Followed by the installation process the retraining of a pre-trained model is depicted. Afterwards, the re-trained model is tested and validated. The chapter ends with the description of the realization of the Android app.

4.1. Dataset

The most famous image dataset is ImageNet which is known from the ILSVRC2012. It contains overall about 14 million Images (**?**). This dataset is also used for nearly all pre-trained models. The dataset includes also a set of 120 different dog breeds with about 150 images each breed. This dog dataset can be downloaded from the following Stanford website `http://vision.stanford.edu/aditya86/ImageNetDogs/`.
Another dataset with dogs is from udacity and can be downloaded from `https://s3-us-west-1.amazonaws.com/udacity-aind/dog-project/dogImages.zip`. It contains 133 dog breeds with a mean of 63 images each breed. The images in this dataset are completely different from the ImageNet dataset.

Because of the reason that the ImageNet dataset is already used to train the pretrained models, the last dataset is used for this project, which is also fully unknown for any model.

4.2. Hardware environment

Different hardware setups are used for this project to compare their performance effects. At first the lowest one is a notebook with a Intel(R) Core(TM) i7-4720HQ CPU @ 2.60GHz and 16GB

memory. The other one is a powerful desktop computer with an Intel(R) Core(TM) i7-6700K CPU @ 4.4 GHz and 16 GB memory. In this computer there is also a NVIDIA GeForce GTX 980 with CUDA capabilities version 5.2. It has 2048 CUDA cores running @ 1.291 GHz and 4 GB memory.
For the mobile application also two different mobile phones are used. The first one is a Samsung Galaxy S4 (GT-I9515) with a Qualcomm Snapdragon S600 Quad-Core @1.9 GHz (32-bit) and 2 GB memory. Android 5.0.1 (API 21) is running. The other one is a Motorola Moto X (2nd Gen, 2014) with a Qualcomm Snapdragon 801 Quad-Core @2.5 GHz (32-bit, ARMv7-ISA) and 2 GB memory. The operating system is Android 6.0 (API 23).

4.3. Installation of software

This chapter describes all necessary steps for installing the software environment including Tensorflow.

4.3.1. Prerequisites

The software environment was set up on the Linux distribution Ubuntu 16.04 LTS. To install the software environment for Tensorflow Python is required. Therefore, the current version of Python 3.6 was installed by default. Tensorflow also supports Bazel which was installed by following command.

```
1 sudo apt-get install openjdk-8-jdk
2
3 echo "deb [arch=amd64] http://storage.googleapis.com/bazel-apt stable jdk1.8" | sudo
   tee /etc/apt/          sources.list.d/bazel.list
4 curl https://bazel.build/bazel-release.pub.gpg | sudo apt-key add -
5
6 sudo apt-get update && sudo apt-get install bazel
7
8 sudo apt-get upgrade bazel
```

Listing 1: Bazel Installation

Futhermore, the package and environment management tool Anaconda was installed by the following steps: First, the Anaconda installer was downloaded from `https://www.anaconda.com/download/#linux`. During the installation process the prompts were answered by the default suggestions except the following prompt: ”Do you wish the installer to prepend the Anaconda3 install location to PATH in your /home/aw/.bashrc ? [yes—no]”. ”yes” was typed in and conda was tested using the ”conda list” command.

In the development of native Android Apps Java is used as the programming language. Within the installation of Anaconda, the JDK in the version 8 was installed.
- CUDA, CUDNN

4.3.2. Tensorflow based on Python

In order to install Tensorflow with Anaconda, an Anaconda environment must be created using following command (refer to listing 2).

```
1 conda create -n tensorflow python=3.6
```

Listing 2: Creating an Anaconda environment

This command depends on the version of Python installed on the computer. Next, the created environment is activated through following line (listing 3).

```
1 source activate tensorflow
```

Listing 3: Activating the Anaconda environment

Futhermore, Tensorflow was installed through the Anaconda environment for CPU only by the following command (refer to listing 4).

```
pip install --ignore-installed --upgrade https://storage.googleapis.com/tensorflow/
 linux/cpu/   tensorflow-1.4.1-cp36-cp36m-linux_x86_64.whl
```

Listing 4: Installing Tensorflow through Anaconda

For GPU the tfBinaryURL was changed to `https://storage.googleapis.com/tensorflow/linux/gpu/tensorflow_gpu-1.4.1-cp36-cp36m-linux_x86_64.whl`.

In order to validate the installation, a following short python script was executed in the anaconda environment (refer to listing 5).

```
# Python
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
```

Listing 5: Installing Tensorflow through Anaconda

If the output is 'Hello, TensorFlow!', the installation was conducted sucessfully. Each time when working with tensorflow, the environment has to be activated using the command in listing 3. As a consequence, the environment has to be deactivated when the work is done using the command 'deactivate'.

4.3.3. Tensorflow based on Bazel

- e.g. Workspace changes for Android SDK, msse4.2

4.3.4. Installing Android Studio and its Delevopment Kit

As an app development environment the free IDE Android Studio in its version 3.0.1 was installed by downloading from `https://developer.android.com/studio/index.html`. After the archive file was extracted, the script studio.sh was executed. Required dependencies are installed by Android Studio itself. Futhermore, Android Studio automatically installs the necessary SDK and NDK. So, the SDK in the version 26.1.1 and the NDK in its version 16.1 were used. If Android Studio doesn't install the necessary Development Kits, the appropriate SDK and NDK version can be installed by selecting the SDK and NDK in the section File - Settings - Appearance and Behaviour - System Settings - Android SDK. Thereby, in order to avoid any collapses when running the app, it's important to check the used tensorflow version in the dependency which has to correspond to the tensorflow version which was used to retrain the model.

4.4. building the models

Alice bis Steps, Andi ab Optimierung, time GPU/CPU -¿ evtl extra subsubsection:
- execution methods -¿ Bazel and Python (incompatible versions)
- Mobilnet -¿ steps, optimierung
- Inception -¿ steps, optimierung
- time related differences of execution
-¿ time CPUs/GPU

4.5. Output Tests and Validation

To test and validate the output of the model regardless of whether the pretrained, retrained or optimized one the python script *label_image* of the 'tensorflow-for-poets'-tutorial is used with

following command. Three variables has to be defined differently which is shown at the beginning of listing **??** depending on the used model and if it already has been retrained or not.

```
#for InceptionV3 models use:
INPUT_SIZE=299       #size of input layer
INPUT_LAYER=Mul      #name of input layer
OUTPUT_LAYER=softmax #name of output layer (only for pretrained model otherwise ' final_result' or as defined in retrain-script)

#for MobileNet models use:
INPUT_SIZE=224       #size of input layer
INPUT_LAYER=input    #name of input layer
OUTPUT_LAYER=MobilenetV1/Predictions/Reshape_1  #name of output layer (only for pretrained model otherwise 'final_result' or as defined in retrain-script)

python -m scripts.label_image \
--graph=${HOME}/retrained_graph.pb \
--labels=${HOME}/retrained_labels.txt \
--output_layer=final_result \
--input_layer=${INPUT_LAYER} \
--image=${HOME}/dl/label_image_pics/Affenpinscher_00001.jpg \
--input_width=${INPUT_SIZE} \
--input_height=${INPUT_SIZE}
```

Listing 6: Call of *label_image.py*

Alternatively with bazel the variables in listing **??** need also to be defined. The call itself is fundamentally the same however the executable binary has to be built before, as shown in listing **??**.

```
bazel build tensorflow/examples/image_retraining:label_image && \
bazel-bin/tensorflow/examples/image_retraining/label_image \
   --graph=${HOME}/retrained_graph.pb \
   --labels=${HOME}/retrained_labels.txt \
   --output_layer=final_result \
   --input_layer=${INPUT_LAYER} \
   --image=${HOME}/dl/label_image_pics/Affenpinscher_00001.jpg \
   --input_width=${INPUT_SIZE} \
   --input_height=${INPUT_SIZE}
```

Listing 7: Build and call of *label_image*

If retraining of the model works fine, *label_image* outputs five labels with the highest and corresponding accuracy which are returned by the model, as shown in following listing **??**.

```
Evaluation time (1-image): 0.350s

001 affenpinscher 0.9940035
038 brussels griffon 0.002003342
042 cairn terrier 0.0008437461
100 lowchen 0.00018292216
099 lhasa apso 0.00012668292
```

Listing 8: Output of *label_image.py*

4.6. Implementation of an native Android App

This subchapter describes the processing of the camera input stream and classifying the particular images. Because of the vast extent of the application, the focus lies on the explanation of how the app works and on its important implementation parts.

When click on the icon of the app, a camera view showing results in the bottom part is loaded. In the background, the ClassifierActivity is set as the launcher activity in the AndroidManifest.xml which is a file containing all configuration settings for the app. When an activity is loaded for the first time, the onCreate method is invoked. Because the ClassifierActivity extends the CameraActivity the onCreate method of the CameraActivity is executed and loads the layout activity_camera.xml consisting of a container and a result field. Next, the setFragment method of the

class CameraActivity is invoked if permission for the camera is given. This causes the replacement of the container with the adapted fragment according to the actual size and orientation of the screen. When the size is set, the method onPreviewSizeChosen is invoked where a TensorFlowImageClassifier is instantiated with following input parameters: MODEL_FILE, LABEL_FILE, INPUT_SIZE, IMAGE_MEAN, IMAGE_STD, INPUT_NAME, OUTPUT_NAME. Details about the Classification follow later in the section.

When the fragment was replaced by the CameraActivity, an instance of the class CameraFragment was created. Next, the method onViewCreated of the class CameraFragment is called and a texture view is created. The latter class is part of the Android API and used for frame capturing from an image stream as OPENGL ES texture. Thereby, the most recent image is stored within a texture object. This object is observed by a listener which invokes the method onSurfaceTextureAvailable when the texture object is available. Within this method the camera is opened by the method openCamera(width, height) given the width and height of the camera preview (**?**). First, the method openCamera sets the camera parameters within setUpCameraOutputs e.g. sensorOrientation, previewSize, cameraId etc. In order to classifiy a given OPENGL ES texture, the coordinate column vectors of the texture must be transformed into the proper sampling location in the streamed texture. So, the matrix has to be prepared with the correct configuration which happens in the method configureTransform. The next important step is done by the camera manager which opens the camera by invoking the method openCamer(cameraId, stateCallback, backgroundHandler). The id represents the specific camera device whereas the stateCallback is necessary to manage the life circle of the camera device and to handle different states of the camera device. The backgroundHandler ensures that the classification is done in background mode. When the camera is opened, the camera preview is started by the method createCameraPreviewSession. Within this method the preview reader is initialized and set to read images from the ImageListener which observes whether an image is available. Therefore, an Imagelistener which was instantiated by the CameraActivity is transfered to the CameraFragment.

When an image is available the method onImageAvailable of the class ClassifierActivity is invoked. The image is read by the preview reader and stored in an object. First, the image is preprocessed meaning transfered into planes, stored as bytes, cropped and stored as a Bitmap which is an Android graphic. Google provides an Tensorlflow Mobile API to run tensors on a performance critical device such as mobile devices. To use this API, the depedency has to be added to the build.gradle file listing 6.

```
allprojects {
   repositories {
      jcenter()
   }
}

dependencies {
   compile 'org.tensorflow:tensorflow-android:+'
}
```

Listing 9: Tensorflow API in build.gradle

When including the API, Tensorflow's class Classifier can be used to classify images. Therefore, the TensorFlowImageClassifier which was initialized before invokes the method recognizeImage(croppedBitmap) on the preprocessed image. Within this method, the image data is transfered from int to float. Then, the previous initialized object of the TensorflowInferenceInterface calls the method feed to pass the float values to the input layer of the model. Afterwards, the inferenceInterface invokes the method run with the specified output layer in order to process the classification. Subsequently, the output data is fetched by calling fetch on the output layer listing 7.

```

 // Copy the input data into TensorFlow.
```

```
3  Trace.beginSection("feed");
4  inferenceInterface.feed(inputName, floatValues, 1, inputSize, inputSize, 3);
5  Trace.endSection();
6
7  // Run the inference call.
8  Trace.beginSection("run");
9  inferenceInterface.run(outputNames, logStats);
10 Trace.endSection();
11
12 // Copy the output Tensor back into the output array.
13 Trace.beginSection("fetch");
14 inferenceInterface.fetch(outputName, outputs);
15 Trace.endSection();
```

Listing 10: Classifying images by the inferenceInterface

Afterwards, the results are reordered according to the highest probability and returned to the result field for displaying the output.

4.7. Deployment and Validation

In order to get the model to run in the application, a few things must be adapted. First, the model in the format of a .pb file and its retrained labels as a text file must be imported. Therefore, a assets folder was created where the model and labels were placed. Next, the path of the model and the labels file must be specified like pictured in listing 8.

```
1  // Input shapes
2  private static final int INPUT_SIZE = 224;
3  private static final int IMAGE_MEAN = 128;
4  private static final float IMAGE_STD = 128.0f;
5
6  // Input Layer
7  private static final String INPUT_NAME = "input";
8
9  // Output Layer
10 private static final String OUTPUT_NAME = "final_result";
11
12 // Model Name from Assets
13 private static final String MODEL_FILE = "file:///android_asset/
   opt4_retrained_dog_graph_mobilenet_0.50_224_700_0.007.pb";
14
15
16 // Label Name from Assets
17 private static final String LABEL_FILE = "file:///android_asset/
   retrained_dog_labels_mobilenet_0.50_224_700_0.007.txt";
```

Listing 11: Including the model in the mobile application

For MobileNet models the default settings are shown in the mentioned listing. Otherwise, if an InceptionV3 is used, the settings must be adapted like shown in listing 9.

```
1  private static final int INPUT_SIZE = 299;
2  private static final int IMAGE_MEAN = 128;
3  private static final float IMAGE_STD = 128.0f;
4  private static final String INPUT_NAME = "Mul:0";
5  private static final String OUTPUT_NAME = "final_result";
```

Listing 12: Setup for InceptionV3

The input size defines the size of the input layer of InceptionV3 which is 'Mul'. Whereas the input layer for MobileNet is specified as 'input'. The IMAGE_MEAN and IMAGE_STD values are needed for converting the image data as integer in float values. The conversion is shown in listing 10.

```
1 for (int i = 0; i < intValues.length; ++i) {
2    final int val = intValues[i];
3    floatValues[i * 3 + 0] = (((val >> 16) & 0xFF) - imageMean) / imageStd;
4    floatValues[i * 3 + 1] = (((val >> 8) & 0xFF) - imageMean) / imageStd;
```

```
    floatValues[i * 3 + 2] = ((val & 0xFF) - imageMean) / imageStd;
  }
```

Listing 13: Conversion of image data integer to float

Futhermore, the output layer has to be specified. For MobileNet and InceptionV3, this was set to 'final_result'.

After setting up the application with appropriate values, the application can be built and run. If the application is running successfully and doesn't terminate, the settings are correct. If the result view contains results, everthing's working. If the results are below 0.1, the threshold in the class TensorflowImageClassifier has to be adapted. It's default setting is 0.01f which means only relevant results higher than 0.01 are shown in the result field. As a consequence, bad results given by the network aren't shown generally. In this case, we optimized the model again instead of lower the threshold. Just for the validation of a correct working app, the threshold was set to 0.001f.

5. Evaluation

Andi
- prio von nierdig zu hoch
- regarding implementation time
- regarding performance
- regarding quality in accuracy
- handy perfomance?

6. Conclusion

- tutorials not complete, different
- which model is better
- Tensorflow Lite conversion failed completely
- prospects, improvements, Recommendations

Beispiele frs referenzieren:
In Figure 9 ist das HS Mnchen Logo zu sehen.

Figure 10: FH-Logo

Oder auch eines Codes wie in listing 11.

```

bottleneck_path_2_bottleneck_values = {}


def create_bottleneck_file(bottleneck_path, image_lists, label_name, index,
                           image_dir, category, sess, jpeg_data_tensor,
                           decoded_image_tensor, resized_input_tensor,
```

```
                           bottleneck_tensor):
  """Create a single bottleneck file."""
  tf.logging.info('Creating bottleneck at ' + bottleneck_path)
  image_path = get_image_path(image_lists, label_name, index,
                              image_dir, category)
  if not gfile.Exists(image_path):
    tf.logging.fatal('File does not exist %s', image_path)
  image_data = gfile.FastGFile(image_path, 'rb').read()
  try:
    bottleneck_values = run_bottleneck_on_image(
        sess, image_data, jpeg_data_tensor, decoded_image_tensor,
        resized_input_tensor, bottleneck_tensor)
  except Exception as e:
    raise RuntimeError('Error during processing file %s (%s)' % (image_path,
                                                                 str(e)))
  bottleneck_string = ','.join(str(x) for x in bottleneck_values)
  with open(bottleneck_path, 'w') as bottleneck_file:
    bottleneck_file.write(bottleneck_string)
```

Listing 14: Some python code

Sectionrefs: In section 2 ist vieles noch nicht fertig.
SubSectionrefs: In section 2.1 wird dann nher auf den Inhalt eingegangen.
SubSubSectionrefs: In section 4.3.2 gehts ans eingemachte.

Beispiele frs zitieren:
Fr einen noch besseren berblick, kann das Buch von **?** oder auch andere refs wie **?** hinzugezogen werden (**?**).
Wenn in klammern und Seitenzahl (**?**, p. 3)

als compared, aber ohne Seitenzahl (cmp. **?**)
als compared mit Seitenzahl, das nd heit ”no date”, da keine Jahrezahl vorhanden (cmp. **?**, p. 5)

List of Figures

A. Anhang

A.1. Anhang vom Anhang0