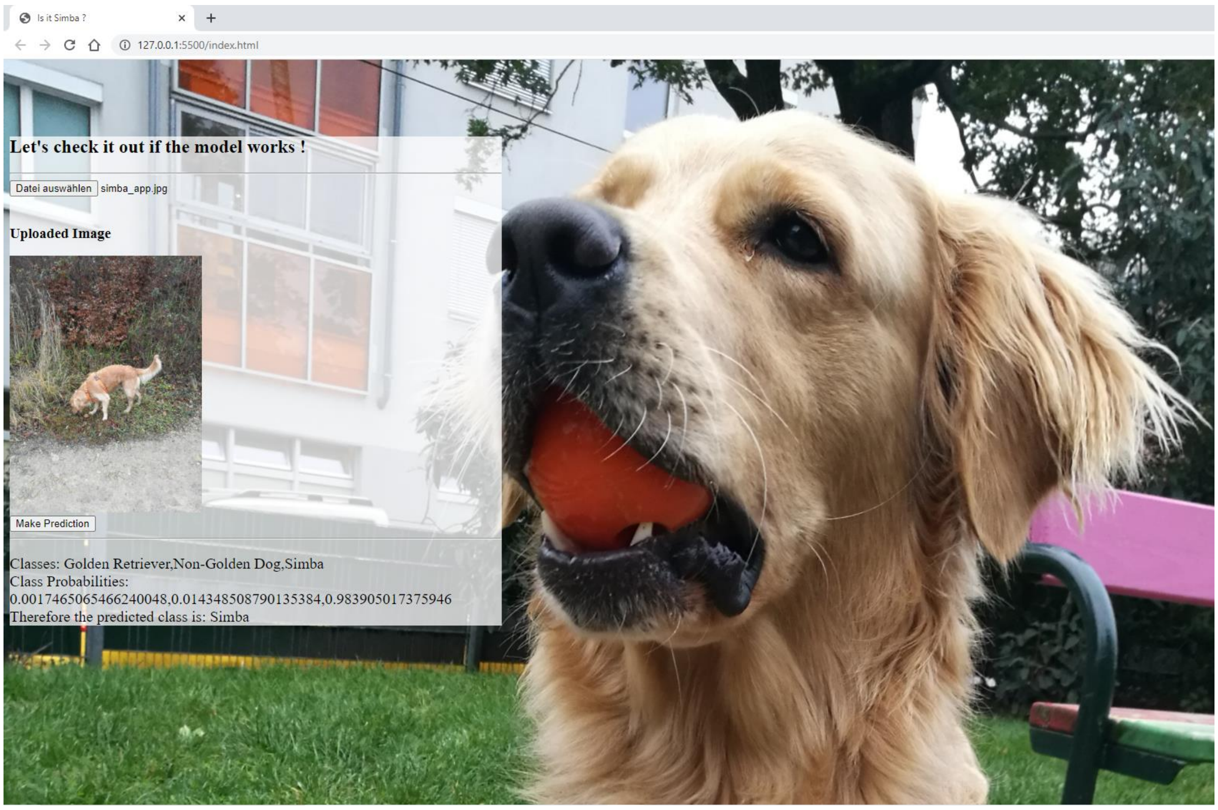
„Is it Simba ?"



Let's check it out if the model works !

simba_app.jpg

Uploaded Image

```
Classes: Golden Retriever,Non-Golden Dog,Simba
Class Probabilities:
0.0017465065466240048,0.014348508790135384,0.983905017375946
Therefore the predicted class is: Simba
```

Author: Christoph Obenhuber, 9851560
Topic: Image Classification
Type: Bring your own data (but also a touch of "Bring your own method")

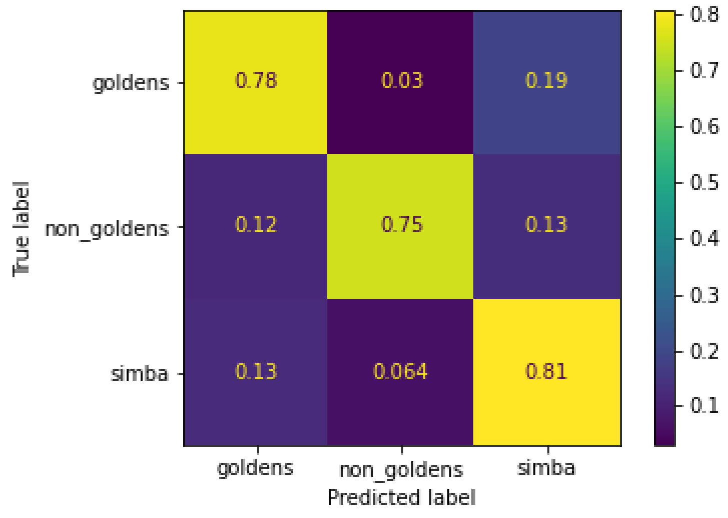
Summary

The screenshot above shows the final outcome of my project, a simple Tensorflow.js web application which predicts the probability of a given (dog) image belonging to one of the three possible classes "Simba" (my dog), "Golden Retriever" (the breed type of my dog) or "Non Golden Dog" (any other dog breed). The user uploads an image and after clicking "Make prediction" the Tensorflow model is loaded in the background, inference is done and the result of the prediction is shown.

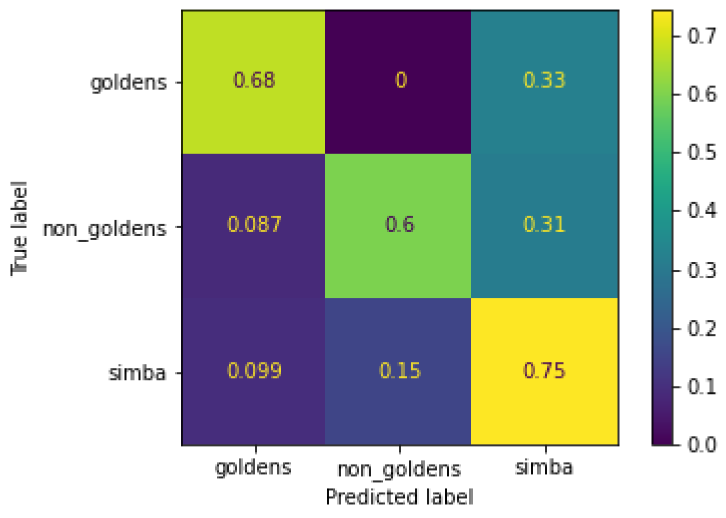
The core of the web application is a convolutional network (VGG3 Style) solving a classification problem for three classes, programmed in Python based on Keras Tensorflow Framework. The dataset used for training and validation of the model consists of ~1000 images per class (I overweighted the proportion for class Simba as the images in this class have much larger variance in their features compared to the other classes). Pictures of Simba were taken by myself on my cell phone, pictures of the remaining two classes are from the "Stanford Dog Dataset" and the "Tsinghua Dog Dataset".

Convolutional neural networks have been widely used in solving image-based problems, such as object/character detection and face recognition, as they allow us to more or less automatically extract a wide range of features from images. There exist many established models (with respect to network architecture and to pretraining) which is why I also selected this approach. Available studies focus primarily on the problems "Cats vs. Dogs" (this is one of the typical introductory projects for image classification with CNNs), "Differentiation between different dog breeds" or "Face recognition of single human beings". Therefore my problem was kind of a combination of the above, trying to identify an individual dog, namely Simba.

The “type” of my project was “Bring your own data” but nevertheless I wanted to gain some insights on feasible network architectures and model tuning based on a simple VGG3 style network (using three stacks of two convolutional layers each). As error metric I chose "Accuracy". Following Cats and Dogs Kaggle competition, it seems to be routine to achieve approximately 80% accuracy with a manually designed convolutional neural network and 90%+ accuracy using transfer learning on this task. As I created my CNN manually, I also chose 80% overall accuracy plus 80% accuracy in group "Simba" as the target I wanted to achieve. Best results on the validation set were 79% overall accuracy and 81% for Simba, so pretty close. Below table shows accuracy for the validation set. The most misclassifications occur between the classes “Simba” and “Golden Retrievers” which makes sense as Simba is also of breed “Golden Retriever”.



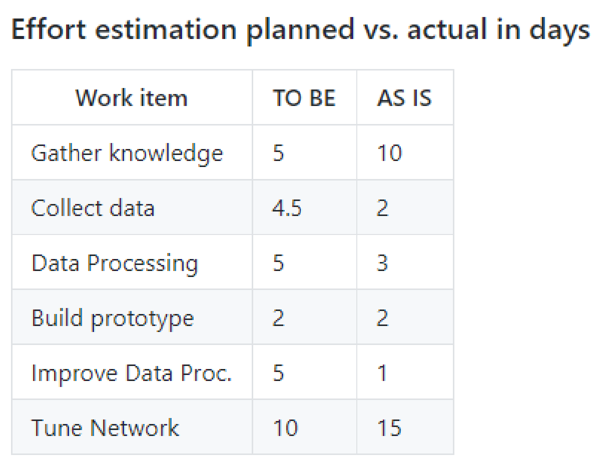
As a final check I applied the model on a test set of so far unseen data in order to check if the model generalized well. Accuracy is not as good as on the validation set but this is expected due to some overfitting during the model training. Also the proportion of class “Simba” is above average which I contribute to the fact that I included proportionally more pictures of Simba than for the other two classes.



Summarizing the project, the chosen convolutional network solved the given problem quite well despite its simple architecture. Due to time constraints not all ideas for tuning and improvement were pursued and many more detailed analysis of results remains open. These will be subject of further work on the topic, among them applying some type of hierarchical

CNN with separate classification layers for breed and Simba yes/no (as proposed by professor Pacha). Generally I am quite happy about the project because I learned a lot and even more important it deepened my interest in the topic of Deep Learning.

Project Plan & Process
The present work was my first deep learning project and also the first time I did an implementation in Python. The same is true for the Keras / Tensorflow framework. This is also reflected in the table above which shows that it took me more time than planned to get used to the new programming language and the specifics of Keras and Tensorflow. Still I was surprised how easy it is to build a first convolutional network with the help of Keras. This is especially true for the data preprocessing and the image loading as well as data augmentation and tuning of the network. Nevertheless the tuning accounted for a large(r) part of the project as the options are kind of "endless". Especially with respect to tuning it is crucial to have a working end-to-end pipeline for loading data, preprocessing, training and evaluating the model. If parts of this chain have to be done manually, a lot of time is lost and the whole process becomes error-prone. If I had to do the project again I would try to come earlier to a working end-to-end pipeline.

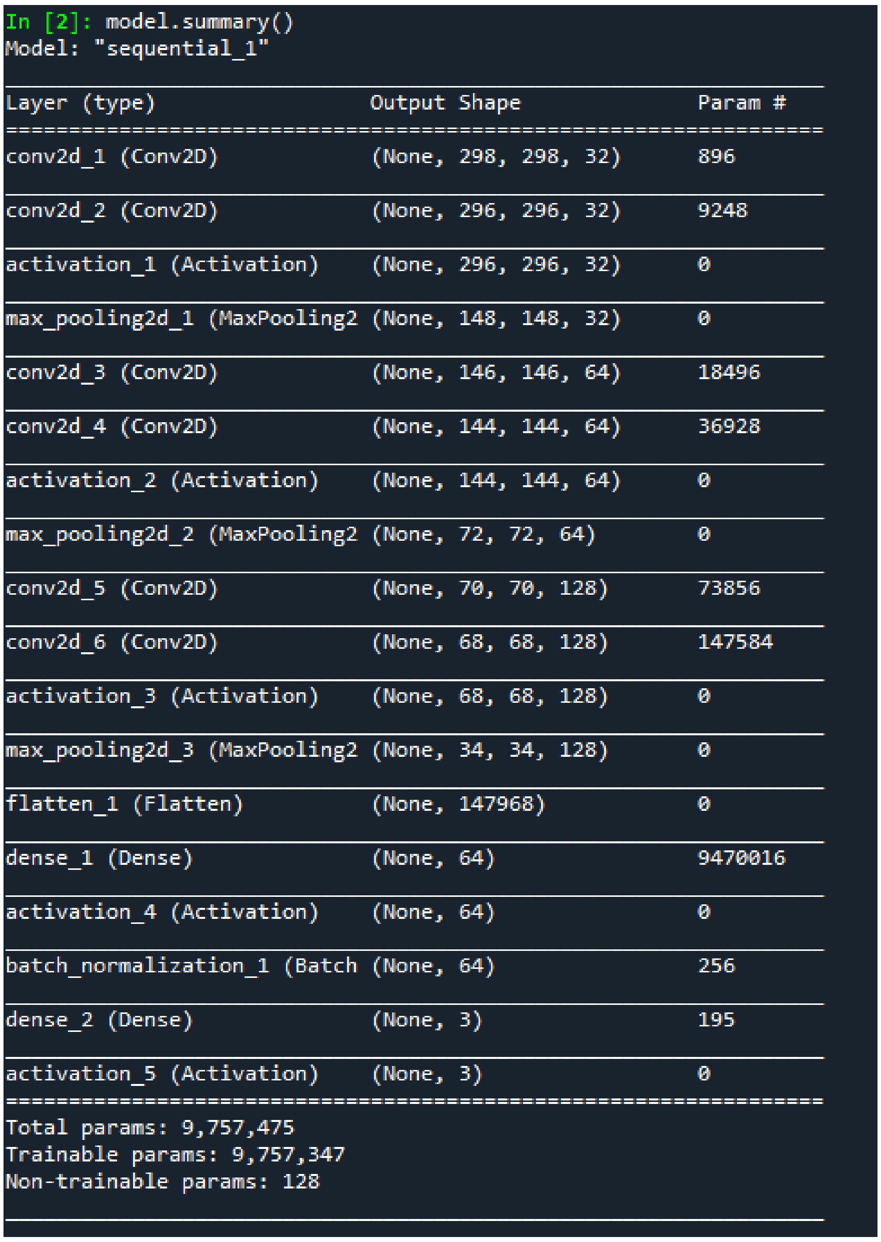


Effort estimation planned vs. actual in days

Work item	TO BE	AS IS
Gather knowledge	5	10
Collect data	4.5	2
Data Processing	5	3
Build prototype	2	2
Improve Data Proc.	5	1
Tune Network	10	15

Dataset, Network Architecture & Tuning
The finally used network architecture and the relevant parameters are the following:

- VGG3 Architecture with 3 Stacked Convolutional Layers with 32, 64 and 128 kernels
- Relu Activations and MaxPooling
- 3x3 Strides
- 2x2 Pool Sizes
- Softmax Activation with BatchNormalization



```
In [2]: model.summary()
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 298, 298, 32)      896       
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 296, 296, 32)      9248      
_________________________________________________________________
activation_1 (Activation)    (None, 296, 296, 32)      0         
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 148, 148, 32)      0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 146, 146, 64)      18496     
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 144, 144, 64)      36928     
_________________________________________________________________
activation_2 (Activation)    (None, 144, 144, 64)      0         
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 72, 72, 64)        0         
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 70, 70, 128)       73856     
_________________________________________________________________
conv2d_6 (Conv2D)            (None, 68, 68, 128)       147584    
_________________________________________________________________
activation_3 (Activation)    (None, 68, 68, 128)       0         
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 34, 34, 128)       0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 147968)            0         
_________________________________________________________________
dense_1 (Dense)              (None, 64)                9470016   
_________________________________________________________________
activation_4 (Activation)    (None, 64)                0         
_________________________________________________________________
batch_normalization_1 (Batch (None, 64)                256       
_________________________________________________________________
dense_2 (Dense)              (None, 3)                 195       
_________________________________________________________________
activation_5 (Activation)    (None, 3)                 0         
=================================================================
Total params: 9,757,475
Trainable params: 9,757,347
Non-trainable params: 128
_________________________________________________________________
```

The Data Set had the following characteristics:

- Class 1 “goldens”: Pictures from Golden Retrievers excluding Simba
- Class 2 “non_goldens”: Pictures from dogs from various breeds excluding Goldens
- Class 3 “simba”: Pictures from Simba (full body of Simba visible)
- Ratio Training vs Validation Samples: 0.7 vs 0.3
- Sample Balance: ~1000 images per class, overweighting of class “Simba”

Data Augmentation used:

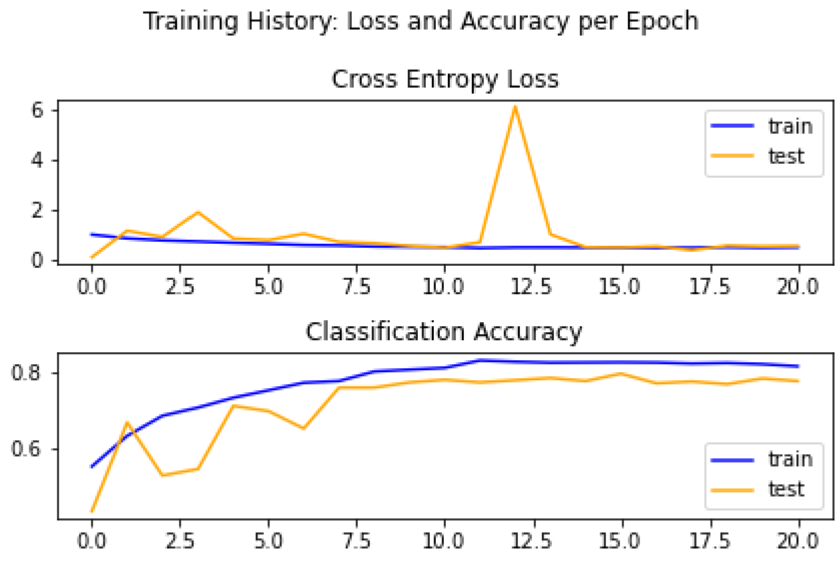
- horizontal_flip=True,
- brightness_range=[0.2,0.8],
- rotation_range=20

Training was controlled by the following parameters:

- Epochs: 30 with early Stopping (which occurred at epoch 21)

- Initial Learning Rate: 0.001
- Learning Rate Scheduler (starting with 0.001 and then getting smaller):
```
# Learning Rate depending on epoch
def lr_on_epoch(epoch_id, lr):
    new_lr = lr - (lr/20.0 * epoch_id)
    return new_lr
```

- Batchsize: 16
- Image Size: 300 x 300
- Optimizer: RMSprop
- Loss Function: Categorical Entropy
- Metric: Accuracy



Final Insights
My main takeaways are the following:

Dataset & Data Augmentation

- In general the number of samples per class should be balanced. If one class is overweighted heavily, the model will strongly overfit on this class and most predictions will be attributed to it.
- Overweighting class "Simba" a bit improved the results. Features of Simba images had larger variance then the ones for the other two categories because pictures of Simba were taken in very heterogenous settings and from different angles whereas the pictures for the other classes are kind of "portrait pictures". Adding some more pictures of Simba "helped" the network to learn more features.
- Slight data augmentation (brightness, a bit of rotation, horizontal flipping) improved the results a bit. Heavy data augmentation (a lot of rotation, zooming, shifting) lead to much worse results.
- Applying data augmentation also to the validation test set (and not only to the training set) improved the results significantly. In the literature it is not quite clear if it should also be applied to the validation set.
- Image Size did not have a big impact (100x100 vs 200x200 vs 300x300)
- Ratio Training vs Validation Samples: as my dataset was quite small I ended up with an optimal ratio of 70/30. I started with 90/10 which lead to very bad generalization (due to the little number of samples in validation set).

Network Architecture

- Keras (based on Tensorflow) enables quick model building
- Using three stacks of two convolutional layers each improved the overall results only slightly compared to three single convolutional layers. But it improved the accuracy when distinguishing Golden Retrievers from Simba.
- Batch Normalization improved the result slightly.
- Adding more kernels did not improve the results after a certain threshold but approaches limits of the GPU of a standalone machine.

Training

- Early Stopping makes absolutely sense in order to save calculation time and avoid overfitting.
- Learning Scheduler improved the results significantly.
- Make sure you have enough data for the validation set.
- RMSprop (Root Mean Square Propagation) as optimizer delivered better results than Adam (Adaptive Moment Optimization).

Model Building Process

- Having a running End2End Pipeline saves a lot of time compared to manually executed steps. Also, it is less error-prone.
- Proper documentation of all interim results and settings saves significant effort later on when drafting the final report about the project. Also, it makes it easier to revert settings to a former status.
- As soon as you have your final baseline model have it checked by a second person to verify that the network / code is properly done. If you start tuning your model based on an erroneous baseline model you will waste a lot of time.
- Run your model on several different sets of input data in order not to produce some overfitted result. Also, on some specific input data even wrong code will produce good outputs by coincidence.
- Check Stack Overflow early if you run into an issue :)