Documentation for the course project „Ausgewählte Kapitel sozialer Webtechnologien“

Hochschule für Technik und Wirtschaft | Fachbereich 4 | Informatik, Kommunikation und Wirtschaft | Studiengang Angewandte Informatik | Wintersemester 19/20

Dozenten: Benjamin Voigt, Steven Mi

Name: Ioakeim Ioakeim (s0564778), Andrej Kazakov (s0564932), Max Wagner (s0563099)

Datum: 14.01.2020

Introduction

“Der Mensch ist, was er isst” said Ludwig Andreas Feuerbach in his essay titled Concerning Spiritualism and Materialism. Food is undoubtedly one of the most integral and important aspects in human life. It shapes our cultures and influences our everyday lives. Nowadays, more and more people are becoming more conscious about their diet, especially as a means of getting healthier, losing weight or eliminating disease. Food image classification is one of the more promising applications in visual object recognition. This means that automatic recognition of food and its nutritional information from images captured using computer vision and machine learning techniques, has become a hot topic in recent years.[1] CNNs are described as “The most successful type of models for image analysis till date”[2] They deliver high accuracy rates with the need of just labeled image data and very little human input. Additionally, people are also relying in smart technologies to help make their lives easier and more practical. Smartphones have shaped and changed our everyday lives and have had a very significant impact in our social interactions. It is not a coincidence that it is called the “Smartphone era”.

Because of this, our original plan was to train a food classifier, with as many as 101 different food categories and combine it with a mobile application. The biggest obstacle we faced with this was that the dataset was very big, with over 100 000 images which translated to total file size of over 5 Gigabytes. Consequently the training was taking far too long, about 12 hours per epoch. As a result, we had to decide whether we would just take a pre-trained model and only retrain the last layer with our own parameters or train our own less complex classifier from the ground up. Since our main goal was to gain equal experience in all aspects of machine learning, we decided to go with the latter. For the new idea, we drew inspiration from Silicon Valley’s fourth episode in season four. The idea is to take a picture of food and then let the application determine if that image is one of a hotdog or not.

Project Objective

The main objective of our project is to implement and train a convolutional neural network for image classification from scratch. We decided to go with iOS as our platform of choice, because between android and iOS, it is the one that we have the least experience with. Inside the application the user can take a picture using the phone’s built-in camera or choose an existing image from the device’s library. The application then uploads this image to an AWS deployed model, where it analyses and classifies it. Finally, the app

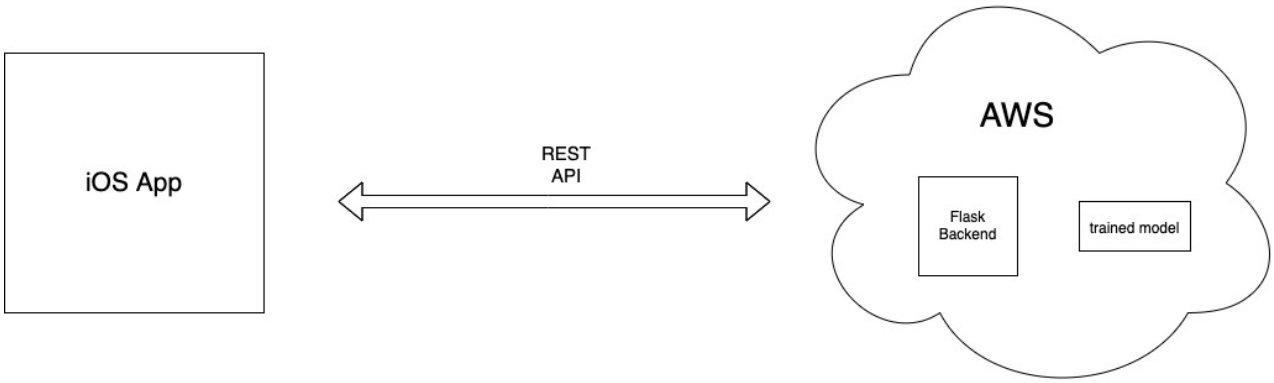
---

[1] see e.g. Kagaya, Aizawa, Ogawa (2014)

[2] Saurabh, Yadav (2018)

responds to the user, whether the image is one of a hotdog or not. In other words, we went for a full stack application for an image classifier.

Throughout this Project, we aimed to learn how to deploy an image classification service onto a server, in this case an Amazon Web Service hosted server. The second goal, was to deepen our knowledge in developing convolutional neural networks from scratch, as was already stated above. Lastly, we wanted to gather experience and understand the challenges that model developing brings. This includes for example, the effect of hyper parameters on the accuracy of the model. The architecture of our finished product, looks like the one in the diagram below:



Methodology

The food classifier dataset, was from kaggle.com. Originally the dataset had no separated images for testing or training, instead it had only images separated into different food categories. There were two text files, a training.txt and a test.txt file, in which the names of every corresponding image were written. So to separate test images from training ones, we wrote a bash script. Among the food categories, there is also a hotdog category so when we switched to the hotdog classifier, we kept the hotdog images and took 3 images from the rest of the categories for testing and 8 for training. We randomly excluded 5 categories from our dataset, to keep the number between hotdog and no hotdog images roughly the same. In the end the structure of our dataset was separated into training and testing folders and in each these we created a hotdog and not hotdog folder. We also renamed all the images based on their category and enumerated them. This allows Keras to better recognise and distinguish between the different labels. We used loss and accuracy as our evaluation strategy. We chose these two because we are familiar with these metrics from the machine learning 1 module last semester. The reason why Keras was our framework of choice, is because it is beginner friendly and uses Tensorflow as backend. For our backend service we went with the web application framework Flask. Flask is lightweight, easy-to-use and it is also designed to make getting started quick and painless. This would allow us to implement a Rest API for our image classification without too much hassle.

Implementation

iOS

At first we developed the iOS app. It is a simple app written in Swift, where we used the Alarmofire networking library to implement our post request to our backend. On the screen the user initially sees only a button. When he clicks on it, he then has the option to take a picture or use an existing one from the phone's gallery. After taking or choosing a picture, the picture is displayed above the button and below the user sees a loading spinner until the backend sends a response. Once the response arrives it displays "hotdog" inside a green background or "not hotdog" inside a red background, depending on the image.

Backend

As a next step, we developed our web service using Flask and deployed it on an AWS server. The development process of the Flask service went smoothly thanks to a guide on the official Keras Blog[3]. Our service has two API endpoints. The first is a get request to simply test whether the server is running. The second is a post request, where the mobile application sends the image to the service for classification. The POST route receives an image, which it pre-processes, in other words it resizes, rescales and transforms it into a numpy array. The output is then passed into our model, where the algorithm predicts whether it is a hotdog or not. Finally it sends the result back to the iOS device.

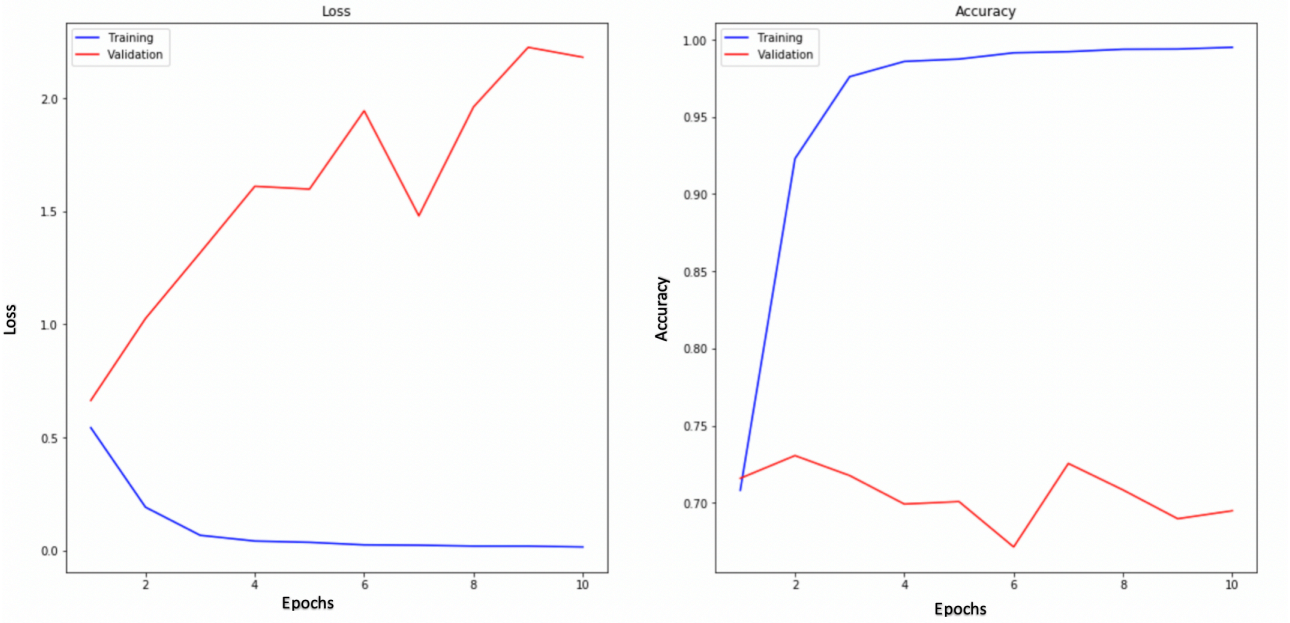
Model

When our backend was finished, we turned all of our attention to the training of our model. We started with two convolutional layers and with 128 dense layers. For optimisation we used the Adam optimiser and binary cross entropy as the loss function. Adam is an optimization algorithm that can be used instead of the classic stochastic gradient descent to update network weights iterative based on training data. The algorithm has two main features: the first one is the adaptive learning rate, which means that the more a parameter is updated the lower the learning rate becomes. The second one is the momentum term, which makes the loss reach the local minimum faster.[4] At first we experimented with 15 epochs, but found out that after 10 epochs the validation accuracy would either stay the same of even get worse. So we settled for 10 epochs. This would also save us a lot of time. Below are the results from our first training:

---

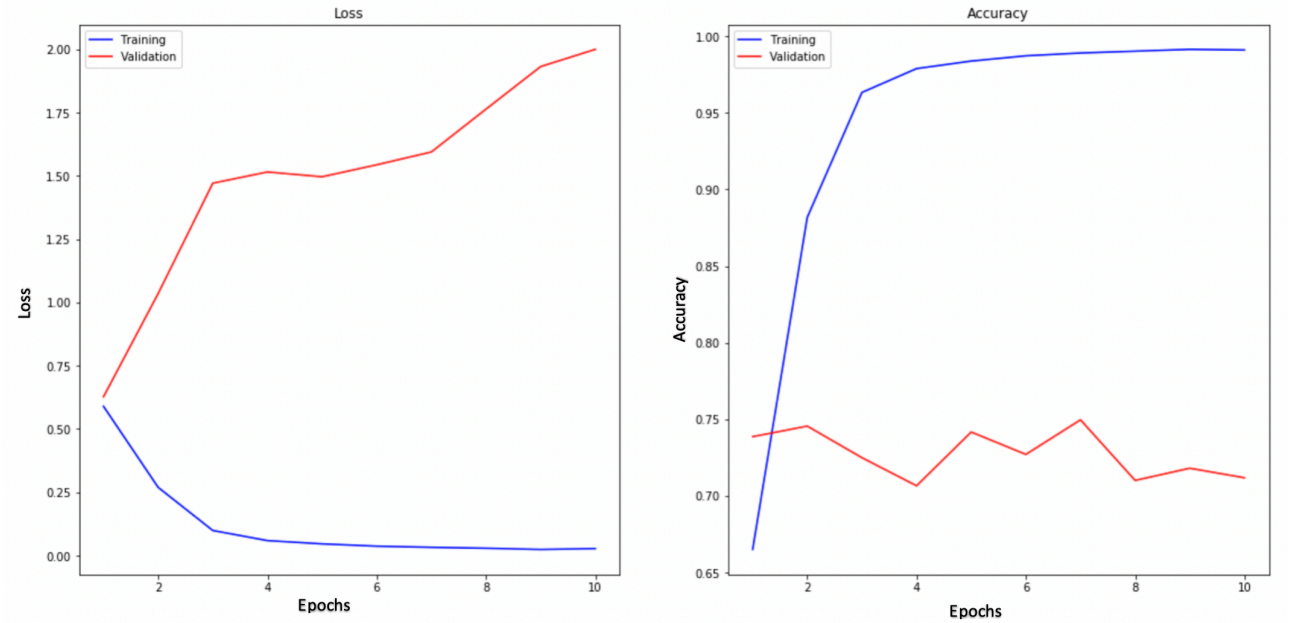
[3] see Rosebrock, Adrian (2018)

[4] see Hansen, Casper (2019)



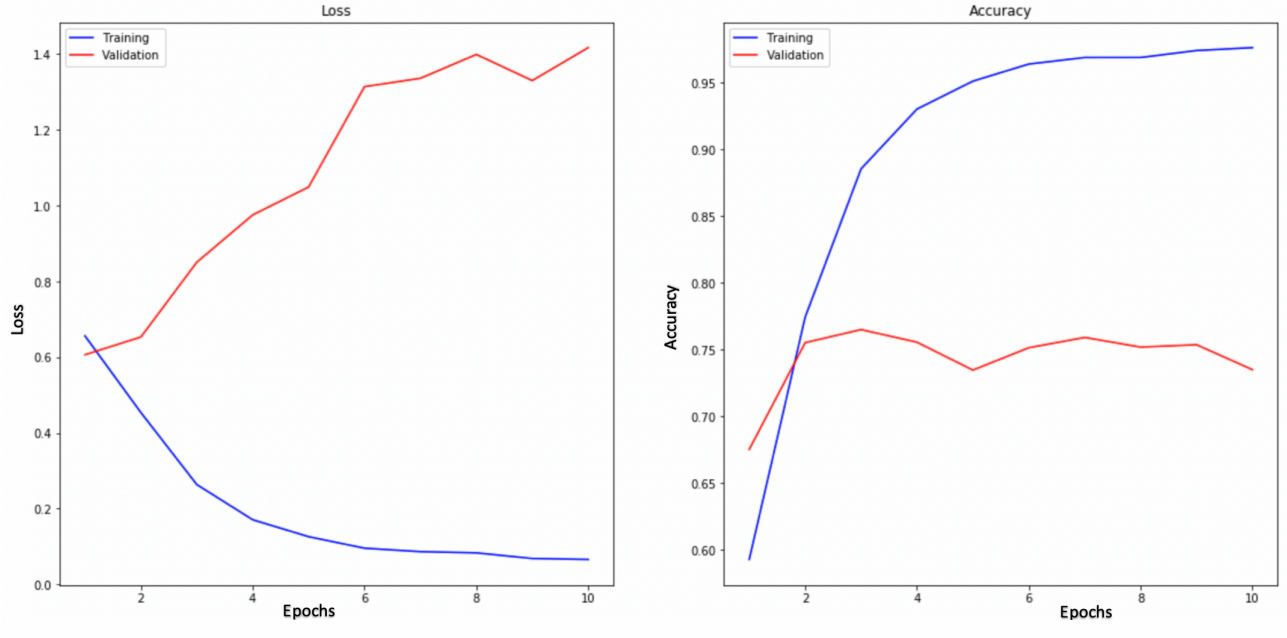
After the first training our validation accuracy was about 70%. Due to the fact that convolutional layers are the major building blocks used in convolutional neural networks, we add another layer as a first step.

Adding a third convolutional layer increases the amount of features that are extracted and thus the image recognition capabilities of our neural networks become better. But of course after a certain number of layers the model will start to overfit the data. In our case adding a fourth layer led to a lower validation accuracy so we stuck with 3 layers. The produced results are shown in the following image:



As seen on the graph above, our training accuracy was about 99% percent. So to avoid overfitting, we decided to add a dropout for the next iteration. At

each training stage, individual nodes are either dropped out of the new with probability 1-p or kept with probability p, so that a reduced network is left. Incoming and outgoing edges to a dropped-out node are also removed. Summarized, dropout is an approach to regularization in neural networks which helps reducing interdependent learning amongst the neurons.[5]



Now we reached a validation accuracy of about 75% and had more consistent results. There are still more techniques to try out, to improve our model. Since the performance of deep learning neural networks often improves with the amount of data available, we wanted to increase the number of our data, specifically our training data. One possibility would be to insert more food images into our training data. We decided to go with data augmentation to achieve this task.

Data augmentation is a technique to artificially produce new training data from existing ones. This is attained by applying domain-specific techniques to examples from the training data that generate new, different training examples.[6]

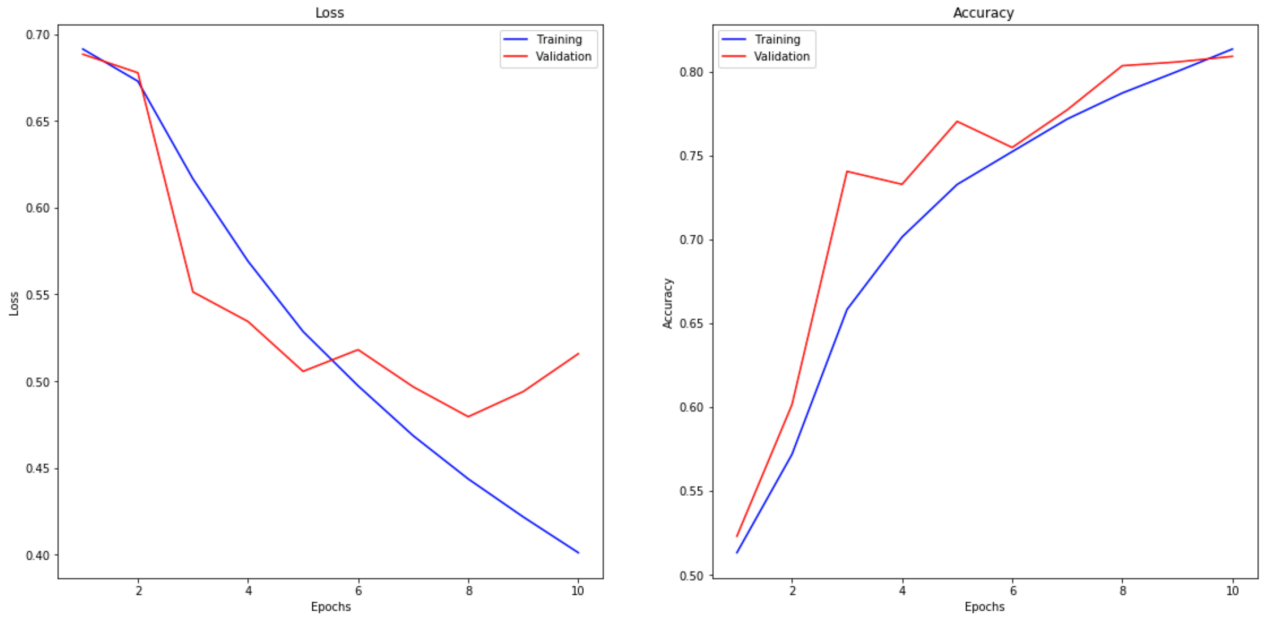
Image data augmentation is one of the most popular types of data augmentation and it has to do with creating transformed versions of images in the training dataset that belong to the same class as the original image[2], for example rotation, brightness shifting or cropped versions of images.

After applying data augmentation we received the following results:

---

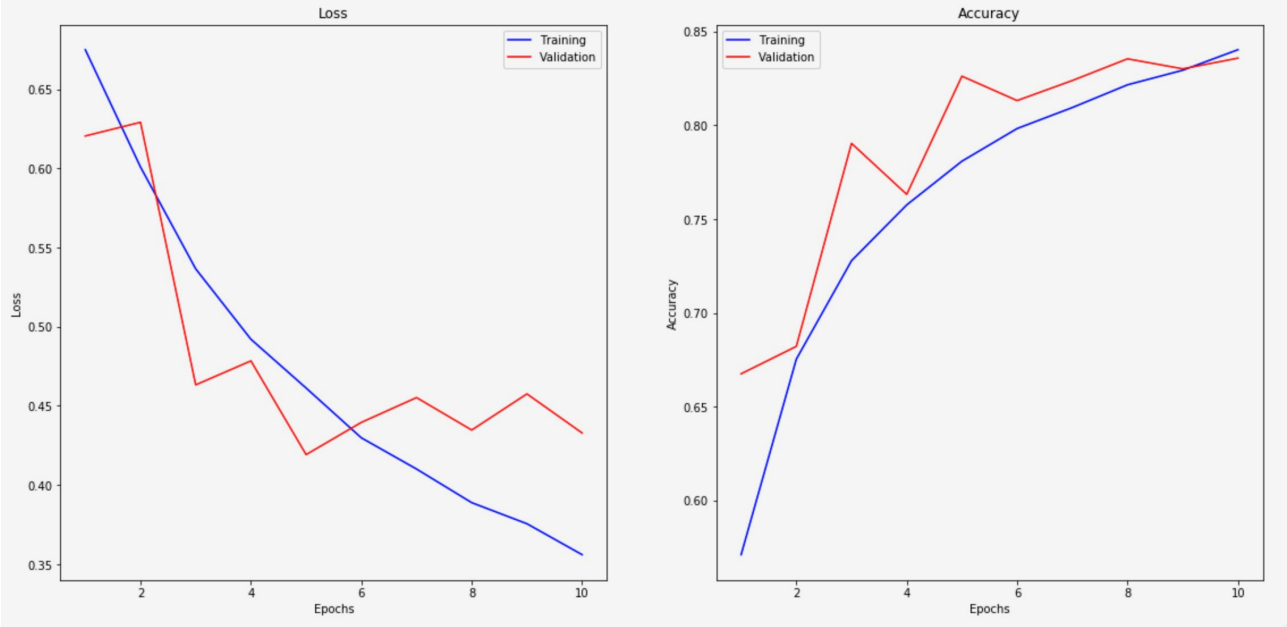
[5] see Budhiraja, Amar (2016)

[6] see Brownlee, Jason (2019)



Finally, we experimented with different optimisers. We found out that the Adadelta optimiser yielded the best results for our model.

Adadelta is a special type of adaptive gradient optimiser(Adagrad). Adagrad uses parameter-specific learning rates, which are adapted relative to how frequently a parameter gets updated during training. With more updates to a parameter, the learning rate decreases. Adadelta tackles this problem by adapting learning rates based on a moving window of gradient updates instead of accumulating all past gradients. This allows Adadelta to continue learning even when many updates have been done, and thus be the superior optimiser between the two.[7] Our results are displayed on the following graphs:



[7] see https://keras.io/optimizers/

Less successful attempts

Although we did not face major difficulties, we still spent some time trying to make our model better which did not always yield the expected results. We tried to play around with the dropout rate, the augmentation rates, the image sizes and shifting some of the test data to the training data. Each of these measures did not improve our model and in some cases even worsened it. Therefore we did not included them in our final model.

Final Results

After multiple training sessions, we managed to achieve a validation accuracy of about 85%. Then we used this model for our backend service and tested it using real life data. As expected we got correct results in about 80% of the time.

We are quite satisfied with the results as it is the first time we have trained a convolutional neural network from scratch by ourselves.

References

[1] Kagaya, Hokuto/Aizawa, Kiyoharu/Ogawa, Makoto (2014): “Food Detection and Recognition Using Convolutional Neural Network”

[2] Yadav, Saurabh (2018): “Brief intro to Medical Image Analysis and Deep Learning”, https://medium.com/@saurabh.yadav919/brief-intro-of-medical-image-analysis-and-deep-learning-810df940d2f7

[3] Rosebrock, Adrian (2018): “Building a simple keras + deep learning REST API”, https://blog.keras.io/building-a-simple-keras-deep-learning-rest-api.html

[4] Hansen, Casper (2019): “Optimizers Explained - Adam, Momentum and stochastic gradient descent”, https://mlfromscratch.com/optimizers-explained/#/

[5] Budhiraja, Amar (2016): “Dropout in (Deep) Machine Learning”, https://medium.com/@amarbudhiraja/https-medium-com-amarbudhiraja-learning-less-to-learn-better-dropout-in-deep-machine-learning-74334da4bfc5

[6] Brownlee, Jason (2019): “How to configure image data augmentation in keras”, https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/

[7] Official Keras Documentation, Optimizers, https://keras.io/optimizers/