Computer Vision Project

Documentation

University of Applied Sciences Vorarlberg

FTB-INF-VZ-19

Artificial Intelligence

Supervised by

Finck Steffen, Hegenbart Sebastian

Submitted by

Ströhle Justin, Öztürk Mert, Nague Marcel, Feistenauer Jakob, Tomasini Nico

Dornbirn, 07.06.2022

Table of Contents

[1. Analysis of Dataset 2](#_Toc105615073)

[2. Data Augmentation and Preprocessing 2](#_Toc105615074)

[3. Used techniques to improve our CNN model 2](#_Toc105615075)

[3.1. Batch Normalization 2](#_Toc105615076)

[3.2. Transposed Convolution 3](#_Toc105615077)

[3.3. Separable Convolution 5](#_Toc105615078)

[3.4. Global Average Pooling 7](#_Toc105615079)

[4. Progress of our CNN model 8](#_Toc105615080)

[4.1. Attempt 1: Very simple CNN 8](#_Toc105615081)

[4.2. Attempt 2: Adding more layers 9](#_Toc105615082)

[4.3. Attempt 3: Adapting number of filters 10](#_Toc105615083)

[4.4. Attempt 4: Best Result 11](#_Toc105615084)

[References 14](#_Toc105615085)

# Analysis of Dataset

We were provided with a set of images of flowers belonging to 102 different categories. It was initially created by two scientists working at Oxford University. According to them, the images were acquired by searching the web and taking pictures. Each class consists of between 40 and 258 images. The official documentation of the dataset can be found using the following link:

<https://www.robots.ox.ac.uk/~vgg/data/flowers/102/>

We received two folders: train and test. The train folder includes 5486 images dedicate for training, while the test folder contains 1351 flowers we should use to test our model in a train-test-split validation style. The CSV files train\_lables and test\_lables contain two columns with image name and respective label (= flower category).

A set of 1352 images from the folder were not included since they will be used to benchmark our final model in the last lecture.

# Data Augmentation and Preprocessing

We used the data augmentation possibility that was provided us through the Data Generator. In addition, we passed a preprocessing function as shown below.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

# Used techniques to improve our CNN model

## 3.1. Batch Normalization

Training Deep Neural Network is a difficult task involving several problems to tackle. One challenge is that the model is updated layer-by-layer backward from the output to the input using an estimate of error that assumes the weights in the layers prior to the current layer are fixed. In practice, however, the layers are updated simultaneously. Batch Normalization (or short Batch Norm) is proposed as a technique to help coordinate the update of multiple layers in the model. Moreover, it improves the learning speed and provides a regularisation mechanism for avoiding overfitting.

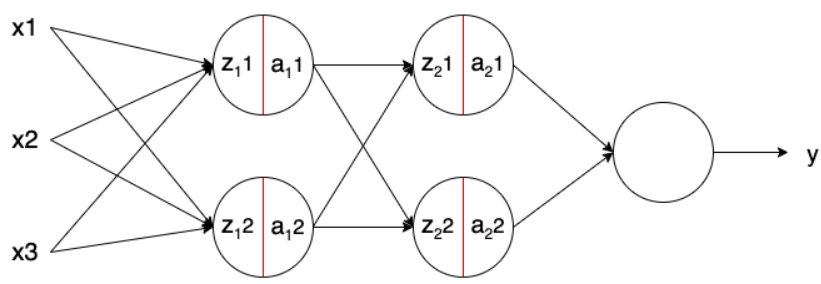
To fully understand how Batch Normalisation works, normalisation needs to be defined. It is a pre-processing method used to standardize data. In other words, having different sources of data inside the same range. Not normalising data before training can cause problems in our network, making it drastically harder to train and decrease its learning speed. One way to normalise data is to force the data points to have a mean of 0 and a standard deviation of 1, using the following formula:

whereby x is the data point to normalise, m the mean of the dataset, s the standard deviation of the dataset. Now, each data point mimics a standard normal distribution. Having all the features on this scale, none of them will have a bias, and therefore, our models will learn better.

Batch Norm uses this way to normalise batches of data inside the network itself. So, the normalisation is done between the layers of a NN instead of in the raw data. Following the formular above, we can define Batch Normalisation formula as:

whereby is the mean and the standard deviation of the neurons’ output.

In the following image, we can see a regular feed-forward Neural Network: are the inputs, the output of the neurons, the output of the activation functions, and the output of the network:



The red line represents the Batch Norm, applied to the neurons’ output just before applying the activation function. Thus, the formula would like:

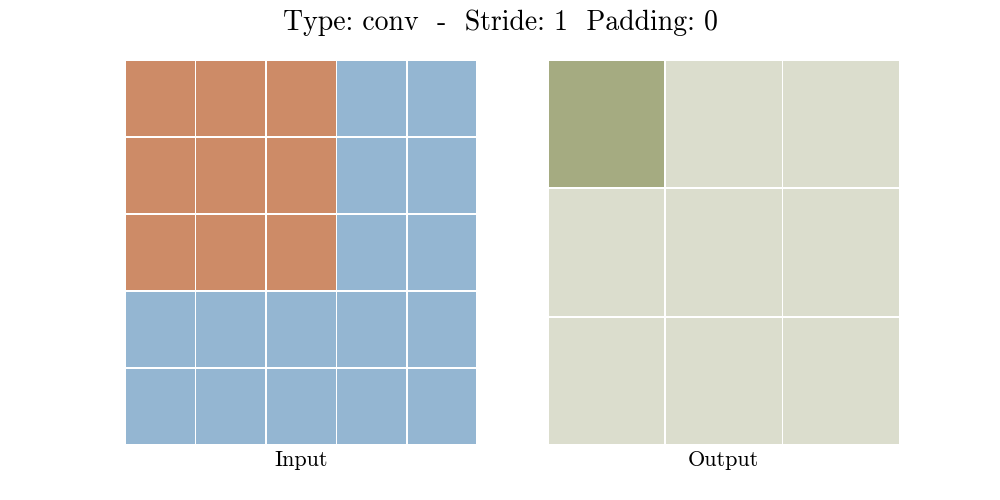
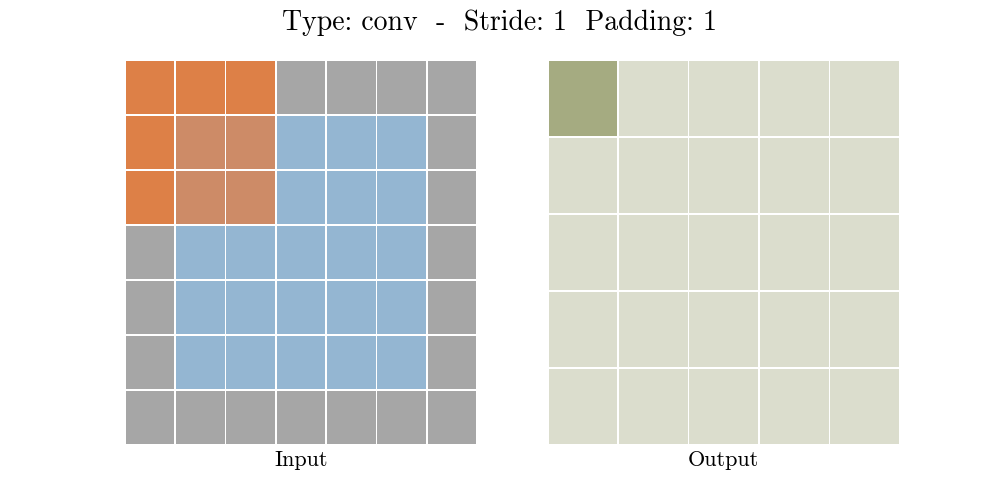
whereby is the output of Batch Norm, the linear transformation of the neuron.

## Transposed Convolution

The idea of a Standard Convolution layer is to do “down-sampling”, meaning that the spatial dimensions of the output are less than that of the input. There are two crucial parameters in this context:

* Padding (p): number of zeros padded around the original input
* Stride (s): amount by which kernel is shifted when sliding across the input image

Firstly, the input image is padded with zeros, while in the second step the kernel is placed on the padded input and slid across generating the output pixels as dot products of the kernel and the overlapped input region (see animation below).

On the other hand, Transposed Convolution is carried out for trainable “up-sampling” i.e. to generate an output feature map that has a spatial dimension greater than that of the input feature map.

Let’s consider a 2x2 encoded feature map to be upsampled to 3x3 by using a 2x2 kernel.

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung Ein Bild, das Shoji, Gebäude, Kreuzworträtsel enthält.

Automatisch generierte Beschreibung

Now we take the upper left element of the input feature map and multiply it with every element of the kernel as shown below.

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

Similarly, we do it the same for the other elements of the input feature map.

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

As it can be seen, some of the elements of the resulting upsampled feature maps are over-lapping. To solve this issue, we simply add the elements of the over-lapping positions, leading to the desired 3x3 output.

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibung

This technique is used for Super-Resolution and Semantic Segmentation. An example of the latter can be found below.

Ein Bild, das Text, Säugetier enthält.

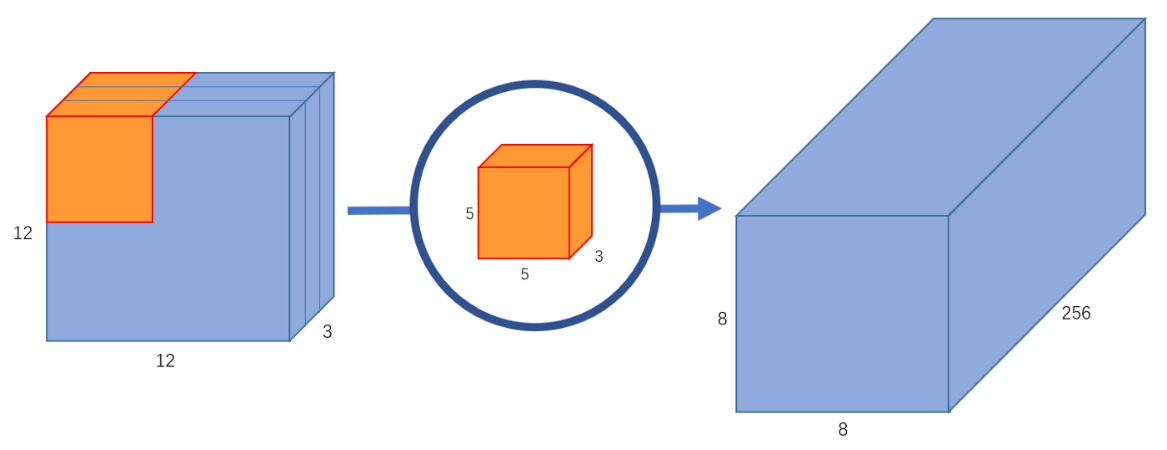
Automatisch generierte Beschreibung

## Separable Convolution

There are two types of Separable Convolution: spatial and depthwise. Since keras.layers.SeparableConv2D uses the depthwise variation, the following explanations concern only this type of Separable Convolution.

It works with kernels that cannot be “factored” into two smaller kernels. It not only deals with the spatial dimensions, but also with the depth dimensions – the number of channels in case of an image. An image may have 3 channels: RGB. A kernel is split into two smaller ones that do two different convolutions: depthwise and pointwise.

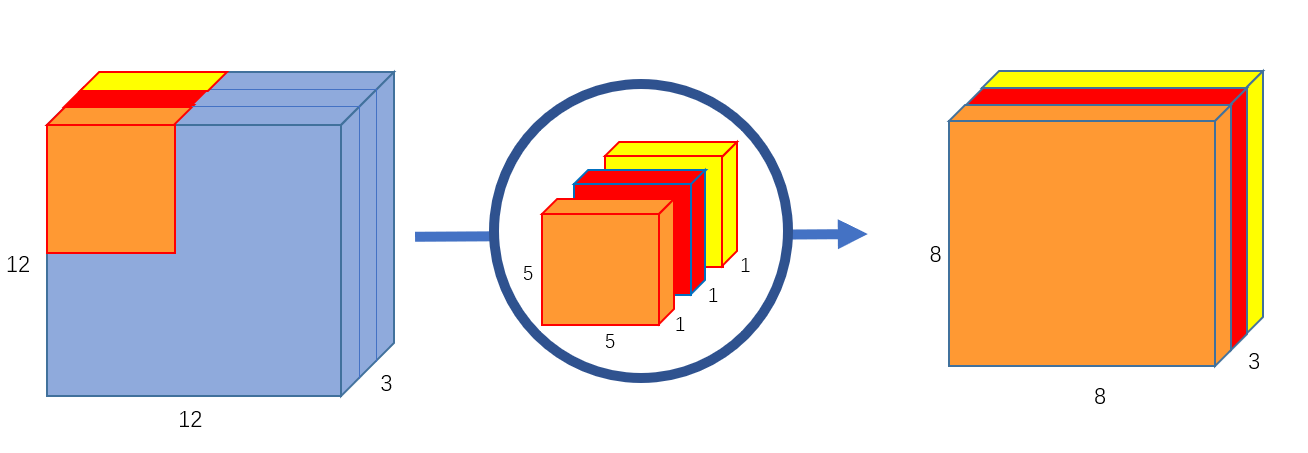
As an example, let’s consider the following scenario. The picture below showcases a normal convolution with an output of 8x8x256.



The goal is now to split the operation into two smaller parts.

*Part 1: Depthwise Convolution*

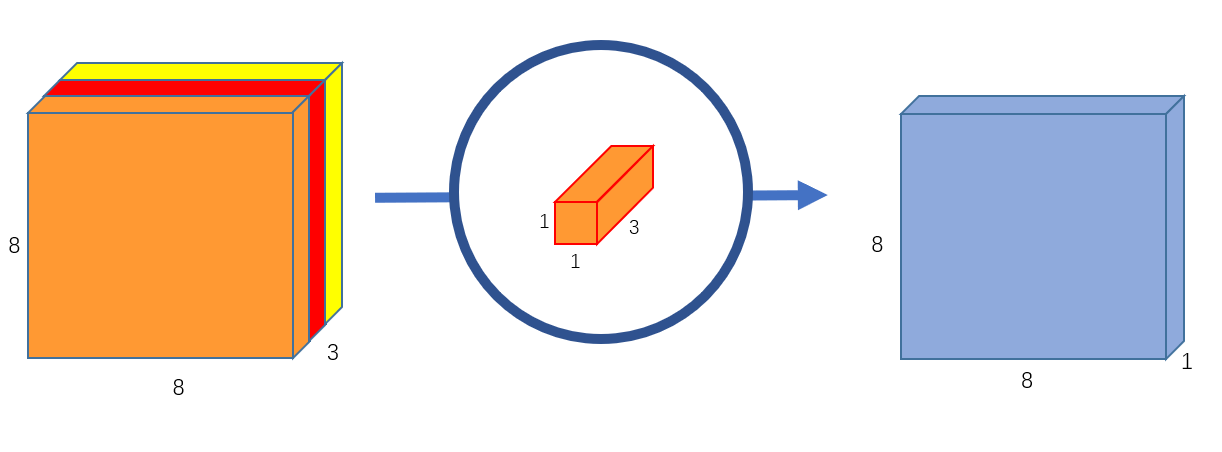
The first part has an input image to which a convolution is done without changing the depth.



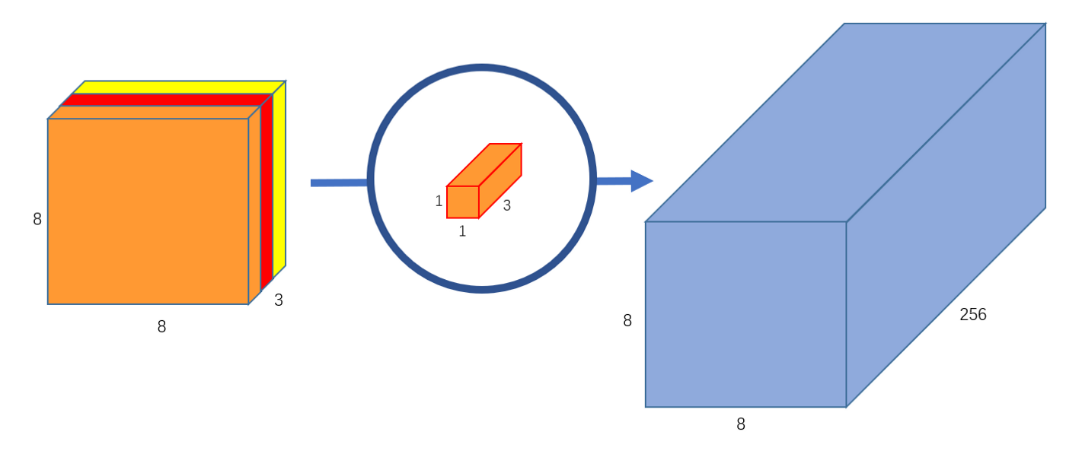
Each 5x5x1 kernel iterates 1 channel of the image (note: 1 channel, not all channels), getting the scalar products of every 25 pixel group, giving out a 8x8x1 image. Stacking these images together creates a 8x8x3 image.

*Part 2: Pointwise Convolution*

Now, we need to increase the number of channels of each image. The pointwise convolution is so named because it uses a 1x1 kernel, or a kernel that iterates through every single point. This kernel has a depth of however many channels the input image has; in our case, 3. Therefore, we iterate a 1x1x3 kernel through our 8x8x3 image, to get an 8x8x1 image



We can create 256 1x1x3 kernels that output a 8x8x1 image each to get a final image of shape 8x8x256.



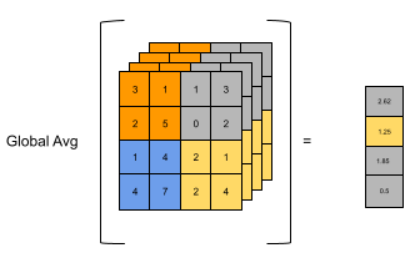
The huge benefit of this approach is that it requires much less computational complexity, resulting in the ability to process more operations in shorter amount of time.

* Number of multiplications in Normal Convolution: 256x3x5x5x8x8=1,228,800
* Number of multiplications in Depthwise + Pointwise Convolution: 256x1x1x3x8x8=49,152 multiplications + 3x5x5x8x8 = 4,800 multiplications à52,952

The number of multiplications is different, but the performed convolution is same in Normal and Depthwise Convolution.

## Global Average Pooling

The feature maps of the last convolutional layer are vectorized and fed into fully connected layers followed by a softmax logistic regression layer. However, the fully-connected layers are prone to overfitting. One strategy to overcome this is to use Global Average Pooling. Usually, it replaces the Flatten layer. It generates one feature map for each corresponding category of the classification task in the last Conv layer.



It takes the average of each feature map, and the resulting vector is fed directly into the softmax layer. This approach comes with two main benefits:

* More native to convolution
* Overfitting is avoided since there are no parameters to optimize

# Progress of our CNN model

## 4.1. Attempt 1: Very simple CNN

Our first idea was to just stick with the techniques we learned from our lecturers, meaning classic convolution layers, max pooling layers, ReLu and Dense layers as well as the learning rate. As our first attempt we decided to go with the easiest approach: Just copy-pasting the model we were shown in the project-introduction session (CNN-Combine Ingredients). So we used three blocks of Conv2D + MaxPool2D layers, with 16 filters of size 3x3.

Ein Bild, das Text enthält.

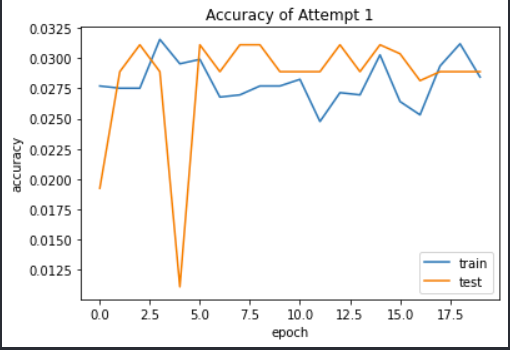
Automatisch generierte Beschreibung

With approx. 2.275.000 parameters, we let our model train for 20 epochs.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

Interestingly, our neural network didn’t really learn anything as it began at 2% and got stuck around 2-3 % train and test accuracy throughout the entire run.



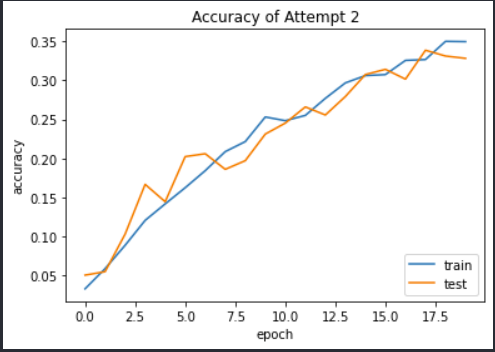
## Attempt 2: Adding more layers

As part of the introductory lesson of the project work, our lectures gave us the hint that in general, it is better to have more layers and less filters in general. We applied this advice by adding three additional blocks of convolution and pooling layers. Apart from that we reduced the learning rate from 0.05 to 0.001. We also kicked out one dense layers since they normally increase the number of parameters a lot. As we wanted, this has led to a significantly less amount of parameters (~ 36.500) compared to attempt 1.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

The trick with adding more layers to our model improved our accuracy by a reasonable amount, reaching 35% for train and 32,8% for test.



In this attempt, the model improved steadily but continuously after each epoch, which was a very positive sign. In CNNs, the first layer usually extracts basic features such as horizontal or diagonal edges. The output is then passed on to the next layer which detects more complex features such as corners or combinational edges. As we move deeper into the network it can identify even more complex features such as objects, faces, patterns etc. This is probably the reason why our model showed regular improvements. Moreover, we didn’t experience any overfitting until this point of time, which is also good.

## Attempt 3: Adapting number of filters

For our dataset, we recognised that 16 is a very small number to choose for the amount of filters. That is why tried increase the number of filters the deeper we went, whereby the kernel size stayed the same (3x3). This is also a common pattern we found in the CNNs mentioned by researchers on the Internet.

Ein Bild, das Text enthält.

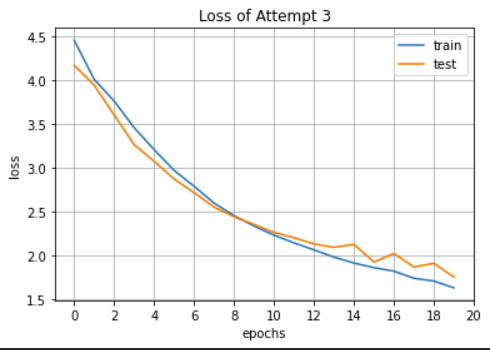
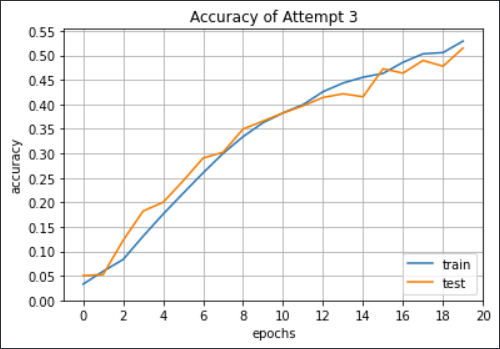
Automatisch generierte Beschreibung

This adaptation obviously gave us more parameters for training, approx. 190.000.

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung

We let model again train for 20 epochs with a batch size of 50, resulting in the following development.



When we put these results in contrast to our previous attempt, we needed less than the half of epochs of attempt 2 to reach the 35%-accuracy mark. This means that our model learned significantly faster, ending up with 53% for train and 51% for test.

## Attempt 4: Best Result

Furthermore, we played around with the number and size of filters a bit more but didn’t receive any better results. So, we basically were stuck at the 55% accuracy mark. At this point, we assumed that we probably won’t improve our model with the little knowledge we had about the different types of layers. That is why we researched some additional possibilities to attain better results.

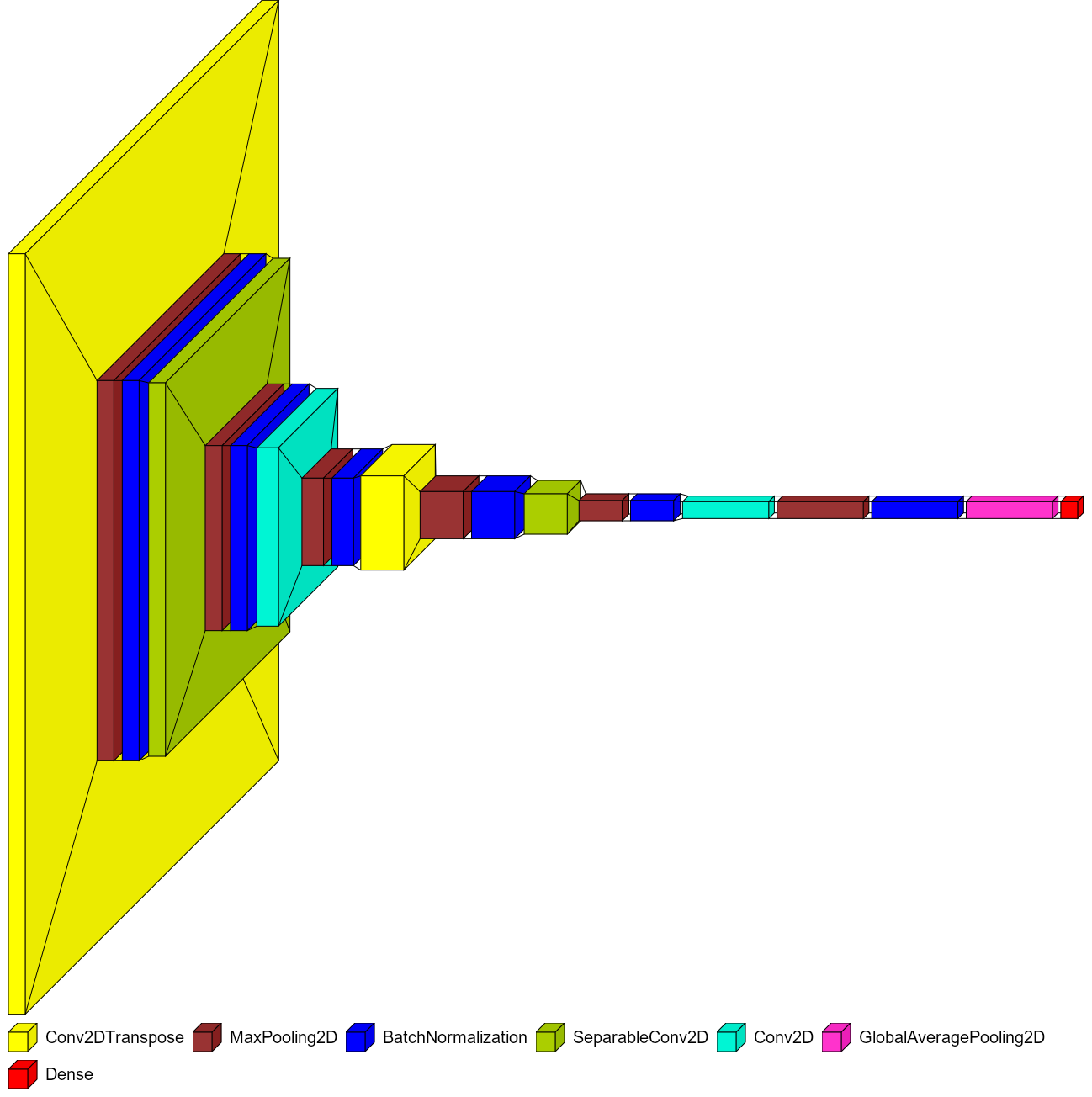
Ein Bild, das Text enthält.

Automatisch generierte BeschreibungThen, we came up with four ways to improve the overall performance of our model: Batch Normalisation, Transposed Convolution, Separable Convolution, Global Average Pooling. The benefits of these techniques were already described in chapter “Used techniques to improve our CNN model”

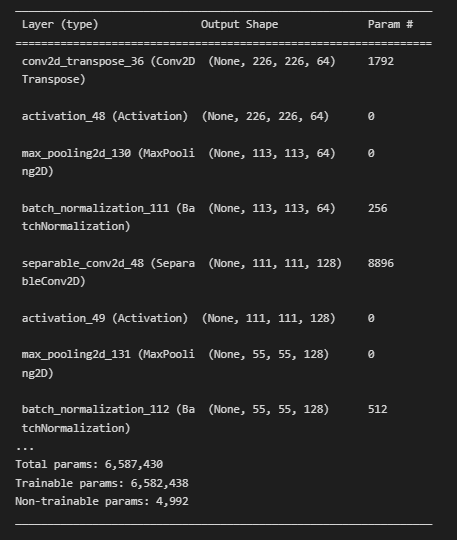
As we can see on the model architecture on the right, we have 6 blocks of different layers. The position of Batch Norm didn’t have any impact on our model even though it is commonly placed between the convolution and pooling layers. We also tried various dense layers (with different units) but came to the conclusion that one softmax-layer with 102 units delivered the best results. Using more than one dense layer has led to more parameters, slower learning speed and partially less accuracy.

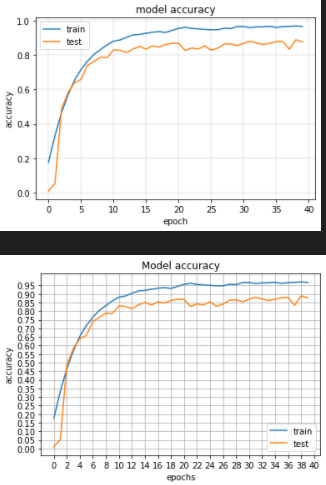
Moreover, the image size didn’t really influence the model. We found out that it is wise to use batch-sizes dividable to the size of the dataset, which is why we chose 26.

The following illustration shows a more appealing visualisation of the model architecture.



The following shows part of the summary of our layers together with the number of total params.





The above plot shows the training process of our best model in 40 epochs. Within the first 10 epochs, it learned very fast, and then started to fluctuate between 84% and 88% for test data. The same goes for the training data, but between 93% and 96%. The best accuracy we achieved in test data was **89.9%**. The time it took to train was about two hours.

# References

<https://towardsdatascience.com/transposed-convolution-demystified-84ca81b4baba>

<https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728>

<https://androidkt.com/explain-pooling-layers-max-pooling-average-pooling-global-average-pooling-and-global-max-pooling/>

<https://www.baeldung.com/cs/batch-normalization-cnn>