# Homework 04

IANNWTF 20/21

Submission until 25th Nov 23:59 via https://forms.gle/H31ckx5251Qg3Ndm6

Welcome back to the fourth homework for IANNWTF. This week we have learned about some more training strategies, specifically optimizers, and we have also learnt about a new architecture: CNNs.

In this homework, we want to build an automatic malaria detection. FYI, malaria is a life-threatening disease with around 225 million cases worldwide each year and it is particularly dangerous for small children.[1] So let's see if we can use our neural networks for a good cause!

*This week we have tried to keep the instructions as short as possible, so that you are challenged to try it yourself. Because the best way to learn something is to do it yourself ;) If you are lost, check the hints for advice or visit us during the Q&A Sessions.*

## 1 Data set

We will work with the Tensorflow Malaria dataset https://www.tensorflow.org/datasets/catalog/malaria. It contains 27.558 coloured images of cells with equal shares of malaria-infected cells and uninfected cells. [1]

Print out some of the images together with their shapes. Do you see a problem here? Do you have an idea how to solve it?[2]

Perform other necessary or beneficial preprocessing steps.[3]

## 2 Model

Implement a Convolutional Neural Network to classify the cells. Just try to combine some layers and see what happens. If you can't make anything work, look at the hints. [4] [5] [6] [7] [8]

[1] Who, World Health Organization:. "Malaria." World Health Organization: WHO, 14 Jan. 2020, www.who.int/news-room/fact-sheets/detail/malaria.

## 3 Training

Then train your network. We would like you to just get started and try it yourself. In the hints we put some training hyperparameters for your orientation in case you're stuck. [9]

For this task, you should reach a test accuracy of 90 - 95%. In the outstanding homeworks we would like to see at least 95% accuracy.

## 4 Visualization

Visualize accuracy and loss for training and test data using matplotlib.

## Notes

[1]Make sure to load the dataset from `tensorflow_datasets` and not keras. We recommend splitting around 80% for training and taking the rest as test data, but there is not just one right solution here.

[2]Problem: the images don't have the same size, but that would be necessary to feed it into our network. Check the Courseware part on Image Representation in the bottom and choose one of the options provided there. Test afterwards if it worked, by printing some pictures and their shapes.

[3] Relevant reprocessing steps:

- Shuffling
- Batching, an orientation would be minibatches of size 64. Feel free to also try other batchsizes and see what happens.
- Normalize the images so your network can work with them better. Check Courseware/Image Representation for inspiration.
- One-hot-encode the labels. What should be the depth of the resulting labels?

[4]You probably want to use convolutional layers: `tf.keras.layers.Conv2D`, kernels of size 3 are always a good choice. The number of filters can vary from something like 20 to over a 100. See what works for you. For activations functions check out 'tf.keras.activations'.

[5]For the first layer, you will need to specify the shape of the input to your CNN. use the flag input_shape for that.

[6]It might be a good idea to alternate convolutional and Max Pooling Layers.
Check out: tf.keras.layers.MaxPool2D(pool_size=)

[7]Use a readout layer. You can decide to reshape your inputs before this read out layer (check out tf.keras.layers.Flatten()) or use global average pooling: tf.keras.layers.GlobalAveragePooling2D()

[8]4 convolutional layers should be enough to reach around 90% accuracy. Feel free to build deeper networks but they need longer to train and are more prone to overfitting.

[9] Training Hyperparameters for orientation:

- epochs: 10 - 25, depending on your architecture you can train even longer as long as the model is not overfitting
- learning rate: should be very small! Try something between 0.00001 and 0.0001.
- optimizer: Adam
- loss: BinaryCrossEntropy. Check `tf.keras.losses.BinaryCrossentropy()`
- accuracy: how many items in prediction and target are the same (in the batched vectors)? $\rightarrow$ take the mean of it