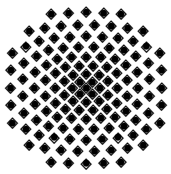

Universität Stuttgart
Institut für Signalverarbeitung und Systemtheorie
Prof. Dr.-Ing. B. Yang

Deep Learning Lab

Project I: Diabetic Retinopathy Detection
Project II: Human Activity Recognition

Winter term 2021/2022

Contents

1 Introduction

1.1 General information

- 6 ECTS credits, 180 hours
  - Time is overall time!
  - 13 weeks with 2 hours presence time
- Team work
  - Two students per team
  - You work together, it's up to you how to do that
  - One report, poster, and presentation per team
- **You have to register the lab this term!** (as any other exam)
  - Online (Campus), Prüfungsamt or Dr. Wizgall (Erasmus, Infotech, . . .)

1.2 Procedure of the lab

- Supervised lab on tuesdays (attendance required)
  - Progress will be checked
- Just working during the lab on tuesdays won't be enough. Most of the work has to be done at home.
- At the end:
  - Scientific paper for one of the projects (3–4 pages)
  - Power pitch for the other project (4 minute presentation + poster session)
  - Deliver your code (make sure that your master/main branch on GitHub is up-to-date)

1.3 Grading

- Working/Proceeding (25 %)
  - Progress, commitment, independence, ideas, teamwork
- Paper (25 %)
  - Do not copy (parts of) script

  - Give an overall picture, but focus on your own contributions
    * Label who did which section
- Power Pitch (25 %)
  - Focus on results and insights
- Code/Result (25 %)
  - Avoid unnecessary code/redundancy
  - Meaningful naming, commenting, spacing, functions, ...

1.4 Infrastructure

Python IDE

We recommend to use PyCharm as your Python IDE. There is a free Community Edition, but as a student you also get a free professional license. Unfortunately, we cannot offer PyCharm on the CIP-Pool computers, you have to use Spyder by using the command `spyder3` within the virtual environment.

GPU-Server

For developing, testing, and debugging use your local machine. For training a model a GPU-Server is provided to deploy your jobs. Further information on how to use it is provided on ILIAS.

Google Colab

For running a longer training you can also use Google Colaboratory https://colab.research.google.com.

A few useful features:

- Mount Google Drive:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

- Clone git repository:
  - Create a new SSH key:

    ```
    ssh-keygen -t rsa -b 4096 -C "your_email@example.com"
    ```

    You will be asked to enter a file in which to save the key, you can press Enter if the default is fine for you. When you're asked to enter a passphrase leave it empty and simply press Enter.
  - Add your public key to GitHub under your personal settings. Under SSH and GPG keys, you can add a new SSH key.

  - Upload your private and public key to Google Colab. The following code snippet allows to upload files from your local computer to Google Colab.

```
from google.colab import files
uploaded = files.upload()
```

  - Run the following commands in Google Colab. In case you named your SSH key differently (not id_rsa), you have to adjust a few things.

```
!rm -rf /root/.ssh/*
!mkdir /root/.ssh
!mv id_rsa /root/.ssh/
!mv id_rsa.pub /root/.ssh/
!echo "Host github.tik.uni-stuttgart.de\n\tHostname
    github.tik.uni-stuttgart.de\n\tIdentityFile ~/.ssh/
    id_rsa" > config
!mv config /root/.ssh/
!ssh-keyscan github.tik.uni-stuttgart.de >> /root/.
    ssh/known_hosts
!chmod 600 /root/.ssh/*
```

  - Set your username and email.

```
!git config --global user.email "stXXXXXX@stud.uni-
    stuttgart.de"
!git config --global user.name "stXXXXXX"
```

  - You can finally clone your repository, as shown exemplary for team 1.

```
!git clone git@github.tik.uni-stuttgart.de:iss/
    dl_lab_19_team1.git
```

- Execute python file:

```
!python main.py
```

- Keep Google Colab alive:
  - For keeping a Google Colab session alive without any interaction, you can run a script in the console which "clicks" every minute. Open the console with `CTRL+Shift+I` and enter the following code line.

```
function ClickConnect(){console.log("Working");
    document.querySelector("colab-toolbar-button#
    connect").click()}
setInterval(ClickConnect,60000)
```

  - Doing that is not intended by Google and may lead temporarily to a suspension of your GPU privileges (happens based on external experience if something is executed for more than 12 hours).

1.5 Best practices

Before starting to write code, we want to give you some best practices which you are supposed to apply throughout the projects.

- Use Python 3 and TensorFlow 2
- Follow PEP 8 (https://www.python.org/dev/peps/pep-0008/), when sensible. PyCharm can automatically check for PEP 8 violations. If you don't want to read the whole style guide, which is understandable, at least follow the naming conventions:
  - Variables, functions, methods, packages, modules: `lower_case_with_underscores`
  - Classes and Exceptions: `CapWords`
  - Protected methods and internal functions: `_single_leading_underscore(self, ...)`
  - Private methods: `__double_leading_underscore(self, ...)`
  - Constants: `ALL_CAPS_WITH_UNDERSCORES`
- Properly structure your repository and code
  - Use modules
  - Object-oriented programming
- Recommendations for Tensorflow 2. Have a look at https://www.tensorflow.org/guide/effective_tf2?hl=de#recommendations_for_idiomatic_tensorflow_20.
- No hard coding! Either use gin (https://github.com/google/gin-config) or write your own config classes.

1.6 Git

For your projects use the provided git repository at https://github.tik.uni-stuttgart.de/. Create separate folders for both projects: `diabetic_retinopathy` and `activity_recognition`. Make sure that your email address in your profile at https://github.tik.uni-stuttgart.de/ and your commit email address in Git are the same, otherwise we can't track your progress. You can set your email address for your repository on your computer with `git config user.email "email@example.com"` (make sure that your current working directory is your local repository). You can find an extensive guide here: https://help.github.com/en/github/setting-up-and-managing-your-github-user-account/setting-your-commit-email-address.

You should use your git repository to only manage source code. **Do not** commit datasets, checkpoints, or results. To achieve that we recommend to use a `.gitignore` file, see https://git-scm.com/docs/gitignore.

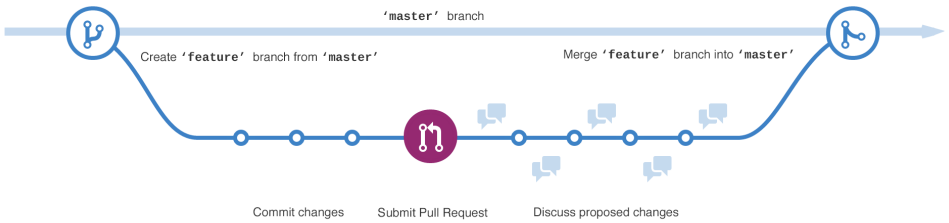

Figure 1.1: Git Workflow (https://guides.github.com/activities/hello-world/)

1.6.1 Git Workflow

We recommend to use the following workflow, illustrated in Figure 1.1, more detailed information can be found at https://guides.github.com/activities/hello-world/.

1. Branch (for every new feature)
2. Commit
3. Pull request
4. Discuss and review
5. Merge

2 Diabetic Retinopathy Detection

2.1 Introduction

Diabetic retinopathy is an eye disease, in which damage occurs to the retina due to diabetes. The longer a person has diabetes and the less controlled the blood sugar, the higher the chances of developing diabetic retinopathy. Diabetic retinopathy is the most prevalent cause of avoidable vision impairment and a major cause of blindness. Because diabetic retinopathy has no early warning signs and can only be effectively treated if it is detected early enough, a regular examination from an eye care professional is very important. Regular screening is very expensive, time consuming and sometimes unreliable. The recognition of diabetic retinopathy is very challenging due to the subtle differences among the individual grades of diabetic retinopathy. Moreover, many small but significant features exist. Therefore, the importance of automatic methods for diabetic retinopathy detection has been recognized.

Diabetic retinopathy is divided into two stages. The first stage is called non-proliferative diabetic retinopathy (NPDR). In this stage, blockages of tiny blood vessels lead to insufficient blood supply. Tiny bulges, so called microaneurysms, protrude from the vessel walls. Sometimes, this causes fluid and blood to leak into the retina. Blurred vision, darkened or distorted sight that is different in both eyes as well as vision loss are typical symptoms in this stage. The only way to detect non-proliferative diabetic retinopathy is by fundus photography or fluorescein angiography. In the former, microaneurysms can be seen. The latter shows narrowing or blocked blood vessels.

The second stage is called proliferative diabetic retinopathy (PDR). In this stage abnormal new blood vessels form at the back of the eye. These can burst and bleed because new blood vessels are very fragile. This causes so called haemorrhages. Specks of blood or spots floating in a person's visual field are typical symptoms in this stage. Fundus photography examination can help to identify cotton wool spots, flame- and dot-blot haemorrhages.

The progress of diabetic retinopathy can be classified into four stages: mild, moderate, severe (non-proliferative) diabetic retinopathy, and the stages of advanced (proliferative) diabetic retinopathy. In the mild stage of NPDR, microaneurysms, small areas in the blood vessels of the retina, begin to swell. In moderate NPDR, multiple microaneurysms and haemorrhages arise, which prevents blood from flowing into the retina. The severe stage of NPDR shows the presence of many new blood vessels. The worst stage of diabetic retinopathy is the proliferative stage. In this stage, fragile new blood vessels and scar tissue form on the retina, leading to blood leaking and finally to permanent vision loss. An example fundus image is illustrated in Figure 2.1.

At present, an eye care professional manually looks for leaking blood vessels, retinal swelling, exudates (pale, fatty deposits on the retina that are signs of leaking blood vessels) and any changes in the blood vessels in fundus photography. Due to the manual nature of diabetic

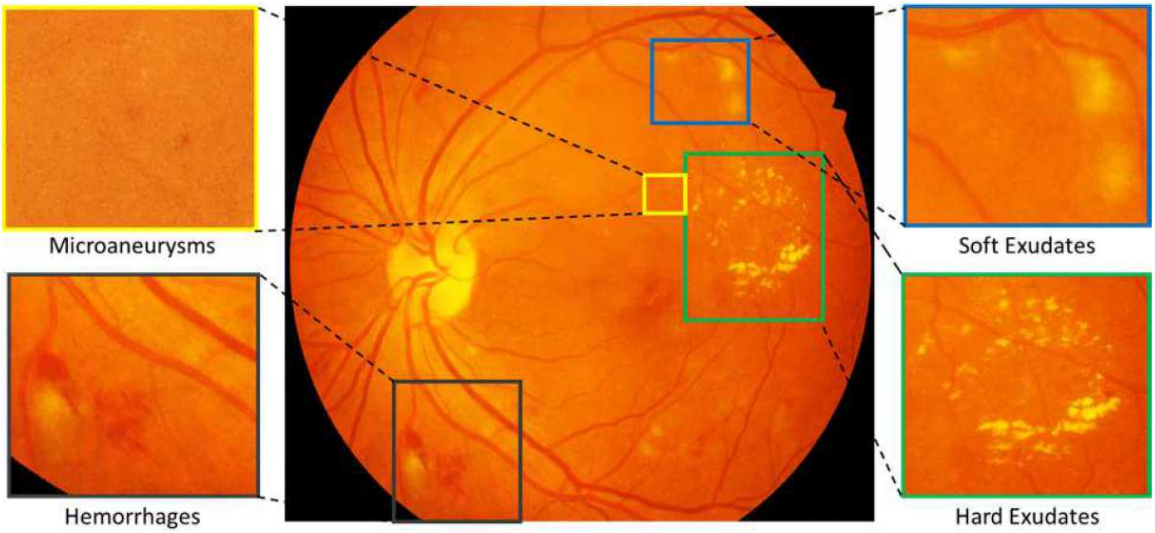

Figure 2.1: Color fundus image showing different symptoms associated with diabetic retinopathy. Enlarged regions highlight the presence of microaneurysms, soft exudates, haemorrhages and hard exudates [1]

retinopathy detection, results are heavily based on the experience of the professional. Therefore results are highly inconsistent [2], thus making automatic diabetic retinopathy diagnosis essential for solving such difficulties.

2.2 Datasets

For the first project use the Indian Diabetic Retinopathy Image Dataset, which is described in the following. It contains relatively few images and makes it therefore quite challenging to obtain high performance. We also offer another optional dataset from a Kaggle challenge to play around with. It contains many and diverse examples.

2.2.1 Indian Diabetic Retinopathy Image Dataset (IDRID)

The Indian Diabetic Retinopathy Image Dataset (IDRID) is a dataset of retinal fundus images that is publicly available. It is a part of the database of "Diabetic Retinopathy: Segmentation and Grading Challenge". Images in the IDRID are categorized into two parts, retinal images with and without signs of diabetic retinopathy. Each image is associated with a ground truth respectively label, that is the severity grade of diabetic retinopathy. The severity grade ranges from 0 (no apparent diabetic retinopathy) to 4 (proliferative diabetic retinopathy) according to the International Clinical Diabetic Retinopathy Scale, see Table 2.1. The dataset is split into a training and test dataset. The database was acquired from an Eye Clinic located in Nanded, (M.S.), India. The database is preselected by experts. Therefore, it mostly contains images with adequate quality and clinical relevance. Moreover, the database does not contain duplicated images and has a “reasonable mixture“ of different stages of diabetic retinopathy

[1]. A summary of the database is displayed in Table 2.2. Example images of IDRID are shown in Figure 2.2.

Label	
0	no apparent DR
1	mild non-proliferative DR
2	moderate non-proliferative DR
3	severe non-proliferative DR
4	proliferative DR

Table 2.1: Labeling according to the International Clinical Diabetic Retinopathy Scale

Total images	516
Image resolution	4288×2848 pixels
Image format	jpg images
Total images in training dataset	413 (80 %)
Total images in test dataset	103 (20 %)
Labels	0...4
Ground truth format	csv files

Table 2.2: Summary of the Indian Diabetic Retinopathy Image Dataset (IDRID)

2.2.2 [Optional] Kaggle Challenge Dataset provided by EyePACS

The Kaggle dataset for Diabetic Retinopathy Detection contains over 80,000 high-resolution retinal images in total. For every subject, an image for the left and right field is provided. Experts have rated the presence of diabetic retinopathy in each image on a scale of 0 (no diabetic retinopathy) to 4 (proliferative diabetic retinopathy), see Table 2.1. The dataset contains abnormal artefacts in images, under- or overexposed images as well as images that are out of focus. All in all, the dataset shows a great variety of different image conditions. A summary of the dataset is given in Table 2.3. Example images of the Kaggle Challenge Dataset are shown in Figure 2.3.

Total images	88702
Total images in training dataset	35126
Total images in test dataset	42670
Total images in validation dataset	10906
Labels	0...4

Table 2.3: Summary of the Kaggle Challenge Dataset

Graham Preprocessing

Due to varying color and brightness levels in the Kaggle dataset, it is recommended to use the preprocessing proposed by Benjamin Graham (winner of the Kaggle challenge). First, the

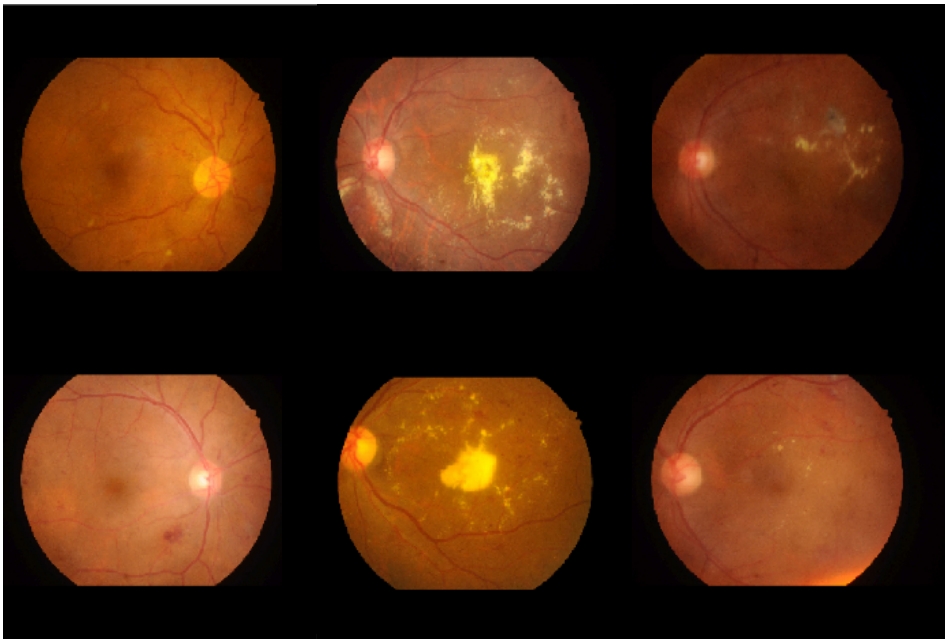

Figure 2.2: Example fundus images of IDRID

images are resized so that the radius of an eyeball is 300 pixels. After that, the local average color is substracted and the local average is mapped to gray. Finally, the images are clipped to have 90 % size to remove the boundary effects. Two examples are shown in Figure 2.4.

If you additionally want to use the preprocessed Kaggle dataset, please ask one of the tutors to provide it.

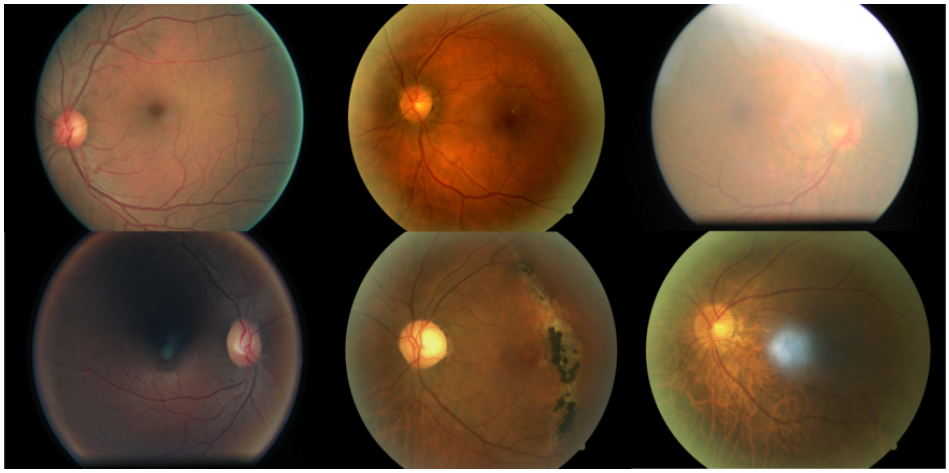

Figure 2.3: Example fundus images of the Kaggle Challenge Dataset

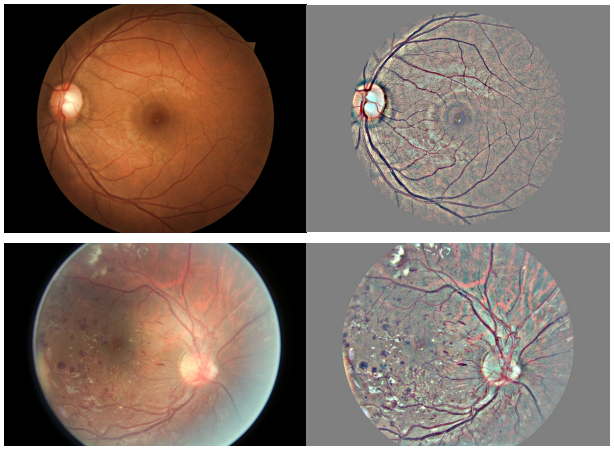

Figure 2.4: Two examples. Original images are on the left, preprocessed images on the right.

2.3 Tasks

Your overall task for the Diabetic Retinopathy Classification is to predict, based on fundus images, whether a patient has non-referable (NRDR) or referable diabetic retionpathy (RDR) (requiring immediate specialist attention). NRDR is considered as having no or mild non-proliferative DR (labels 0 and 1). RDR is considered as having moderate, severe, or proliferative DR (labels 2 and up).

Build a generic framework for your deep learning pipeline step-by-step as described in the following sections. Use at least one deep visualization method to get insights into your trained model.

2.3.1 Input pipeline

Use `tf.data` to build an efficient data input pipeline. For a good performance with `tf.data` have a look at the following guide: https://www.tensorflow.org/guide/data_performance?hl=de. We want to especially highlight the **Best practice summary** section. For a

guide regarding the `tf.data` API have a look at https://www.tensorflow.org/guide/data?hl=de.

For the IDRID dataset, a train and test split already exists. Split the training set into a train and validation set for potential hyperparameter tuning. **Resize the images to a resolution of** 256×256 **pixels without distorting them.** The `tf.image` API offers various useful operations for working with images (https://www.tensorflow.org/api_docs/python/tf/image?hl=de).

Make sure to check your class distribution. For highly imbalanced datasets neural networks tend to learn to predict the class with the higher prior probability. One way to address this problem is balancing your dataset by resampling.

In general, there are two ways to load your images. Either directly from raw data or from TFRecord files. The former can be quite inefficient, especially when you have to read lots of small files. Serializing your data and storing it in a set of files (100–200 MB each) allows reading data efficiently. This is especially the case when reading data from HDDs or streaming it over a network. Decide for yourself if you want to spend time creating TFRecord files, if you want to do it, have a look at https://www.tensorflow.org/tutorials/load_data/tfrecord?hl=de.

Task: Implement an efficient data input pipeline for the IDRID dataset. Make sure to create a train, validation, and test split for potential hyperparameter tuning.

[Optional] Profiling

If you want to analyze the performance of your input pipeline or even your overall training performance, TensorFlow offers a Profiler. The results can be visualized in TensorBoard and can give insights into how you can tune your pipeline to decrease the required time for a training step.

Figure 2.5: Profiler example from https://www.tensorflow.org/tensorboard/tensorboard_profiling_keras

An example output is shown in Figure 2.5. It shows that the GPU idles while waiting for the next input batch, prefetching would help to decrease the training time in this case.

If you're interested in profiling your pipeline, you can start looking at the guide https://www.tensorflow.org/tensorboard/tensorboard_profiling_keras and ask a tutor for further help.

2.3.2 Model architecture

Choosing a good model for a certain task and dataset can be challenging. The following questions might help to find suitable architectures.

- How much data do I have?
- How complex is my data?
- Do I have any time constraints in production?
- ...

In principle, using well-known architectures is always a good idea, an overview for object detection architectures is illustrated in Figure 2.6. Each model is also categorized by its feature extractor, which can give you a grasp of the performance as a classifier.

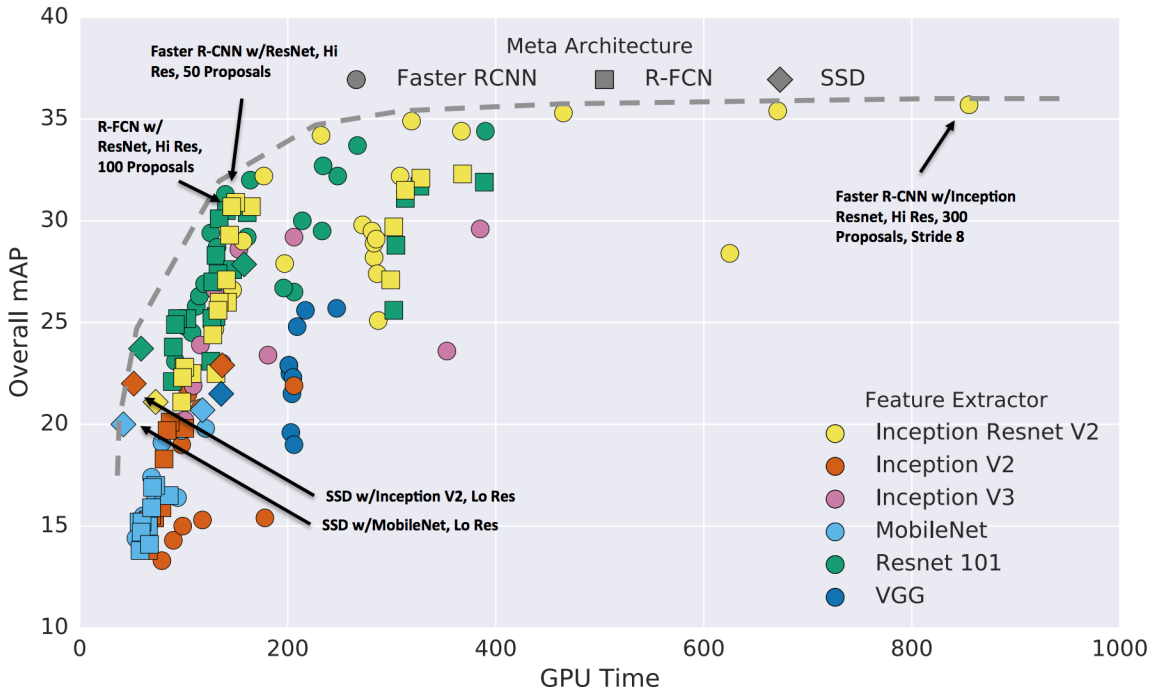

Figure 2.6: Mean average precision vs. time for well-known object detection models on the COCO dataset. [3]

Using such big models might not be the best idea for a dataset as small as IDRID, you might want to think about your own architecture. Of course you could use transfer learning to overcome the problem of overfitting, which can be addressed in the following Task 2.3.7.

Task: Think of a reasonable architecture and implement it using the `tf.keras.Model` class (https://www.tensorflow.org/api_docs/python/tf/keras/Model) in combination with `tf.keras.layers` (https://www.tensorflow.org/api_docs/python/tf/keras/layers). Print the model summary and check the number of parameters within each layer.

2.3.3 Metrics

To evaluate the performance of your model, you have to define reasonable metrics. It is your task to find metrics suitable to evaluate your results and making them comparable to results of others. Thus, you should use common metrics that are widely used in the field of Diabetic Retinopathy classification. Moreover, you may additionally need some metrics that reflect the nature of your dataset, e.g. if your dataset is highly imbalanced. Possible metrics are:

- Confusion matrix
- (Balanced) Accuracy
- Sensitivity
- Specificity
- ROC/AUC
- ...

For implementing any metric, use the `tf.keras.metrics.Metric` class https://www.tensorflow.org/api_docs/python/tf/keras/metrics/Metric?hl=de.

2.3.4 Training and evaluation

Training routine

We recommend to write your own training routine instead of using the `fit` method offered by `tf.keras.Model`. It gives you a better understanding of what actually happens and in particular it provides more flexibility. The guide https://www.tensorflow.org/tutorials/quickstart/advanced tells you how to write your own training and evaluation loops from scratch. If something is unclear, we encourage to ask one of the tutors.

Our recommendation for an optimizer is `tf.keras.optimizers.Adam` (https://www.tensorflow.org/api_docs/python/tf/keras/optimizers/Adam), but if you want to use a different optimizer feel free to do so.

> **Task:** Implement a training routine that regularly logs your progress on the training and validation data.

TensorBoard

TensorBoard is a tool to keep track of your training progress. Have a look at https://www.tensorflow.org/tensorboard/get_started#using_tensorboard_with_other_methods and `tf.summary` (https://www.tensorflow.org/api_docs/python/tf/summary) to log your training progress.

TensorBoard also offers a dashboard for hyperparameter tuning. For more information read https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams.

In case you want to use TensorBoard within your Google Colab notebook, have a look at https://www.tensorflow.org/tensorboard/tensorboard_in_notebooks.

Task: Use TensorBoard to visualize the training progress. TensorBoard is a great way to quickly analyze runs and compare them to others.

Saving and loading checkpoints

There are two ways to save a TensorFlow model. You can save a checkpoint, which only captures all parameters of your model without saving a description of the computations defined by the model. On the other hand, the SavedModel format includes a serialized description of the computation defined by the model in addition to the parameter values (checkpoint). Models in this format are independent of the source code that created the model. It's up to you which format to use, but we encourage you to have a look at both possibilities. More information about checkpoints can be found at https://www.tensorflow.org/guide/checkpoint and for the SavedModel format have a look at https://www.tensorflow.org/guide/saved_model.

Task: Save your model regularly during training so that you do not lose too much progress in case of any failure. Moreover, implement a functionality which allows you to continue training from a saved checkpoint/model.

Evaluation

Task: Implement an evaluation method that loads a specified checkpoint and evaluates your model on the test set.

2.3.5 Data augmentation

Due to the size of the IDRID dataset, you may observe overfitting. Thus, data augmentation is required to increase model performance. For image classification tasks, there are many different augmentation operations. Some of the most common operations are:

- Rotation
- Flipping
- Cropping
- Shearing
- ...

It is up to you to choose suitable data augmentation operations. You can also implement several different augmentation operations and try to use combinations of those to avoid overfitting as far as possible.

Task: Extend your data input pipeline with several data augmentation operations. Apply data augmentation online which means that you create new data on the fly during training. Analyze to which extent a data augmentation operation has an effect on the performance.

2.3.6 Deep visualization

Even though there has been tremendous progress and breakthroughs of Deep Learning in a variety of computer vision tasks, such as image classification, object detection, semantic segmentation, image captioning, and more, there is still a lack of methods which allows to decompose these networks into intuitive and understandable components. As a result, when the deep neural network fails, they fail "leaving a user staring at an incoherent output, wondering why." [4]

"In order to build trust in intelligent systems and move towards their meaningful integration into our everyday lives, it is clear that we must build transparent models that explain why they predict what they predict." [4]

This is where deep visualization methods come into play. They can help to identify failure modes, thereby helping researchers focus their efforts on the most fruitful research directions. Moreover, after deployment of a model, they help to build trust and confidence for a user. And last but not least, when the model is significantly stronger than humans (e.g. chess or Go [5]), it can teach a human how to make better decisions. [4]

Visualizing activations and weights

One straightforward way of visualizing what a network has learned, is to visualize the activations for an image or simply visualizing the weights.

Visualizing the weights of a convolutional neural network helps in the sense that well-trained networks usually learn smooth filters without any noisy patterns. Noisy patterns may be an indication for overfitting. Even though the first layer, as shown in Figure 2.7 for a trained AlexNet [6], shows some general interpretable filters, filters at higher layers are less interpretable. Therefore, directly visualizing network filters isn't practical.

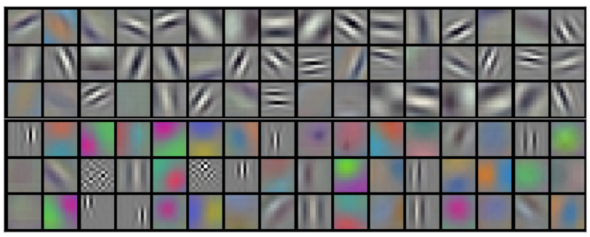

Figure 2.7: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer for AlexNet trained on ImageNet. [6]

Dimensionality Reduction

Another totally different approach is that CNNs can be interpreted as gradually transforming the input image into a representation where the classes are linearly separable (second-last fully-connected layer, before the final prediction layer). Applying dimension reduction methods, such as PCA, t-SNE [7], or UMAP [8], to the embedded images gives a rough idea about the topology of this space. Visualizing the images in the reduced feature space shows which

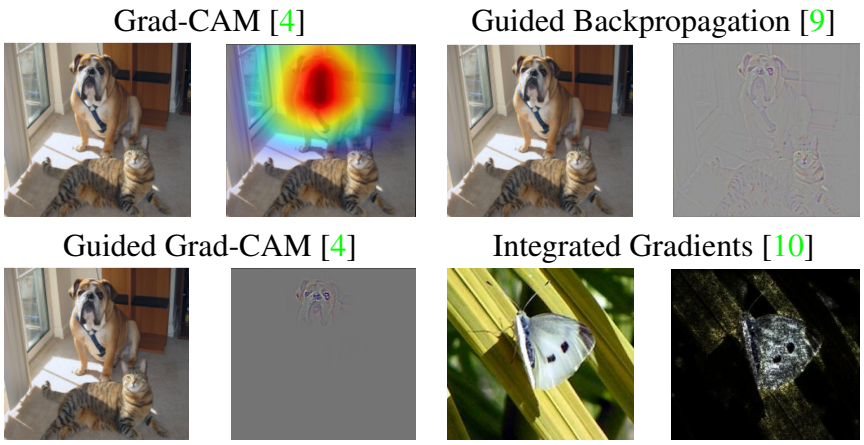

Table 2.4: Examples of the different visualization methods for ImageNet

images are nearby and therefore close in the network's representation space, implying that they are similar.

TensorBoard provides the previosuly mentioned dimensionality reduction methods out of the box. Unfortunately it's not straightforward to use these in TensorFlow 2. In case you're interested, please talk to one of the tutors.

Gradient-based methods

In this lab we want to consider some more sophisticated and more interpretable deep visualization methods which are all more or less gradient-based.

[4] defines a "good" visualization method to satisfy the following properties:

- **Class-discriminative** (i.e. localize the target category in an image).
- **High-resolution** (i.e. capture fine-grained detail).

Answer for yourself which of the following methods are class-discriminative and which are high-resolution.

In the following, we describe some of the most common deep visualization methods: Grad-CAM [4], Guided Backpropagation [9], Guided Grad-CAM [4] and Integrated Gradients [10]. If you want to get a deep understanding, we highly encourage you to read the corresponding papers. Figure 2.4 shows an example for the previously mentioned visualization methods. Further visualizations can be found in the original papers.

Task: Implement at least one of the following deep visualization methods. Implement them on your own, do not use any given libraries. Use the deep visualization method to gain deeper insights into your trained model(s) and potentially detect failure modes.

Gradient-weighted Class Activation Mapping

Gradient-weighted Class Activation Mapping (Grad-CAM) provides coarse localization maps that highlight important regions in an image for a specific prediction. Grad-CAM makes ex-

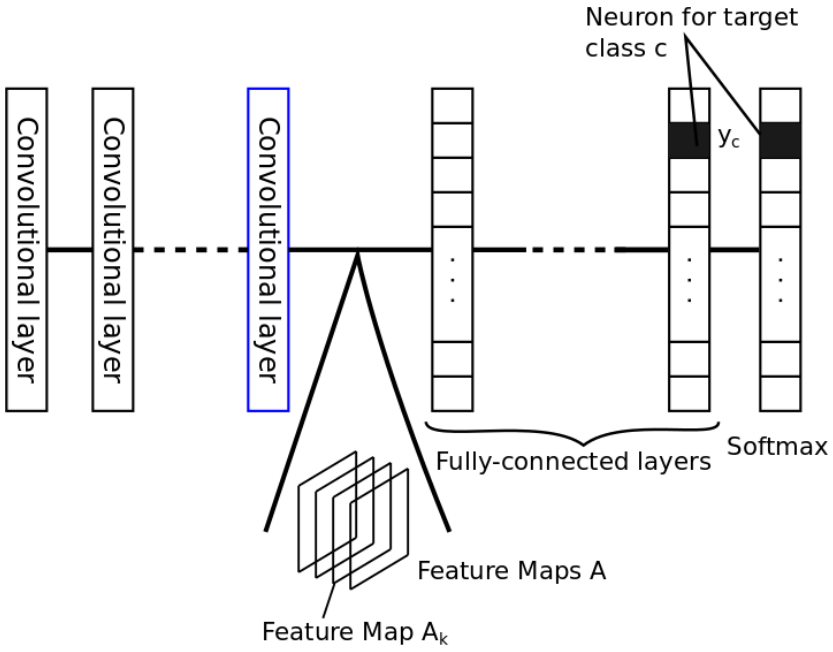

Figure 2.8: Illustration of Grad-CAM

isting state-of-the-art deep models interpretable without any changes in their architecture and therefore is applicable to a wide variety of (convolutional) neural networks. [4]

In order to obtain the localization map Grad-CAM $L^c_{Grad-CAM} \in \mathbb{R}^{u \times v}$ of width u and height v for any class c, the gradient of the logit y^c (output before the softmax) for class c, is computed with respect to feature maps A^k of the last convolutional layer (best compromise between high-level semantics and detailed spatial information). These gradients flowing back are global-average-pooled to obtain the neuron's importance weights α^c_k, which capture the "importance" of feature map k for target class c. This is summarized in the following equation and illustrated in Figure 2.8 [4]:

$$\alpha^c_k = \overbrace{\frac{1}{Z}\sum_{i=1}^{u}\sum_{j=1}^{v}}^{\text{global average pooling}} \underbrace{\frac{\partial y^c}{\partial A^k_{ij}}}_{\text{gradients via backprop}} . \tag{2.1}$$

A weighted combination of forward activation maps, followed by a ReLU, gives the Grad-CAM, as follows:

$$L^c_{\text{Grad-CAM}} = \text{ReLU}\Big(\sum_k \alpha^c_k A^k\Big). \tag{2.2}$$

ReLU is applied after the linear combination of maps because we are only interested in the features that have a positive influence on the class of interest. Negative pixels are likely to belong to other categories in the image. [4]

Guided Backpropagation

Guided Backpropagation is a high resolution visualization technique (pixel-space gradient visualization technique) that visualizes the parts of an image that are most discriminative for a given unit in a network. Guided Backpropagation enables accurate, recognizable reconstructions that visualize the concepts learned by deeper layers in the network. It is applicable to intermediate layers as well as to the last layers of a network. [9]

To visualize the part of an image that activates a given neuron most, Guided Backpropagation computes the gradient of the activation with respect to the image. In addition, only positive gradients are propagated in the backward pass to add an additional guidance signal. This prevents backward flow of negative gradients, corresponding to the neurons which decrease the activation of the higher layer unit we aim to visualize. [9]

Hint: For a network which uses only ReLU activations after each layer, one has to simply override the backward pass of the ReLU function, to mask out negative gradients. Since we don't use any activation after the input, we have to mask out negative gradients that are back-propagated to the input in addition. Writing a custom gradient operation works as follows:

```
@tf.custom_gradient
def guided_relu(x):
    y = # forward pass
    def grad(dy):
        return # custom gradient
    return y, grad
```

Guided Grad-CAM

Guided Grad-CAM is a combination of Grad-CAM and Guided Backpropagation. Important regions of an image that correspond to a certain decision are visualized in high resolution even if the image contains multiple possible concepts (e.g. contains both a dog and a cat corresponding to separate classes).

For calculating Guided Grad-CAM, Grad-CAM and Guided Backpropagation are simply combined via elementwise multiplication. This requires upsampling $L^c_{\text{Grad-CAM}}$ to the input image resolution using bi-linear interpolation.

Integrated Gradients

Integrated Gradients [10] attributes the prediction of a deep network to its inputs. Compared to the previous methods, it has a strong theoretical justification, so in case you want to use Integrated Gradients, we highly encourage to read the paper. Section 5 (Applying Integrated Gradients) of the paper describes how to implement the method.

2.3.7 [Optional] Transfer learning

Training a deep neural network containing millions of parameters can be tough if the dataset at hand is small. Since having a dataset of sufficient size is rather the exception than the

default case, it is quite common to use transfer learning, which aims to transfer knowledge from a first task, called source, to a second (similar) task, called target. This can improve the training process of the target predictive function or the function itself.

The most common way to achieve such a knowledge transfer is by pre-training the network on a very large source dataset (like ImageNet) and then using the final weights as a starting point for the second training phase called fine-tuning where the actual target task is solved. If we take for example ImageNet as a source, the model learns more generic features and suffers less from overfitting. These generic features may then also be useful for the target task [11] and contain more information than the random initialization.

Since usually not only the number of classes but also the classes themselves differ between the two tasks, the output layer must be replaced by a task-specific one and can then only be initialized at random. The need for such a replacement is even clearer if one considers going from classification to semantic segmentation.

While the first publications using such an approach completely froze the pretrained layers [11, 12], this can lead to an inferior performance in cases where there is a noticeable distance between the source and target tasks/domains, like when knowledge is transferred to the medical domain [13]. In such a case, the features extracted from ImageNet may significantly differ and need to be adapted to the new dataset by fine tuning (all) the layers of the model [13].

While the authors of the aforementioned paper sequentially add more and more layers to the fine-tuning process (starting with the last one), it is also possible to fine tune all of them at once, which is more common in practice. In order to prevent large deviations from the initialization, the learning rate used for fine-tuning pre-trained layers is typically smaller than the one used for randomly initialized ones (like the output layer). Although it was shown that fine-tuning can have a positive impact on the performance [14], the gain diminishes as the distance between the tasks and the number of target samples increases [13].

For a deeper insight into transferability of features in deep neural networks, we recommend to read [14]. [15] gives some more insights into understanding transfer learning for medical imaging.

There are two different ways how to use pretrained models in TensorFlow: Using TF Hub (https://www.tensorflow.org/tutorials/images/transfer_learning_with_hub) or `tf.keras.applications` (https://www.tensorflow.org/tutorials/images/transfer_learning). We recommend to use `tf.keras.applications`, an overview of available models can be found at https://www.tensorflow.org/api_docs/python/tf/keras/applications?hl=en. Make sure to use the corresponding preprocessing functions for your input.

2.3.8 [Optional] Ensemble learning

Ensemble learning uses model averaging to reduce generalization error. “The reason that model averaging works is that different models will usually not make all the same errors on the test set.“ [16]

There are various ways how ensemble methods construct the ensemble of models. One can use different subsets of the training set, different model architectures/hyperparameters, or just

different random initializations for training an individual model.

Combining multiple trained models can be done by averaging the prediction or in the case of classification voting can be applied as well. Usually there is no significant benefit from using more than five models.

2.3.9 [Optional] Multi-class classification

Instead of simply considering the binary classification problem of differentiating between non-referable and referable diabetic retinopathy, you can try to classify all stages according to the International Clinical Diabetic Retinopathy Scale (Table 2.1). Often, for medical tasks there are no clear boundaries between classes, so regression might be more suitable.

3 Human Activity Recognition

3.1 Introduction

Human Activity Recognition (HAR) is a problem that is an active research field in pervasive computing. An HAR system has the main goal of analyzing human activities by observing and interpreting ongoing events successfully. Such a system can either use visual or non-visual (sensory) data (Fig. 3.1).

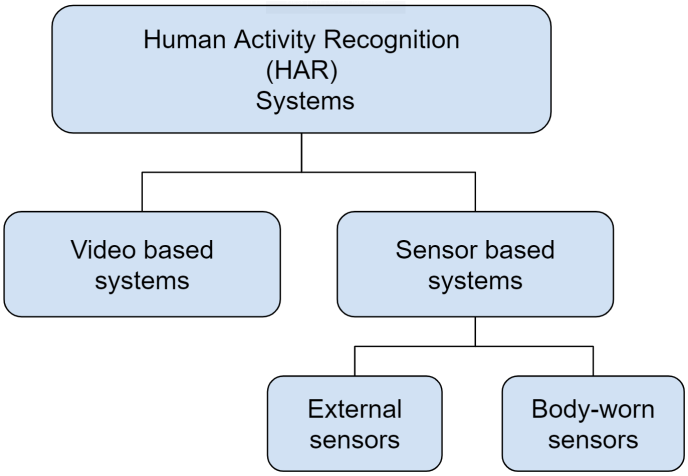

Figure 3.1: Classification of HAR systems

Visual systems typically use one or multiple cameras to monitor human activities. Such systems are deployed in applications such as surveillance for behavioural analysis to identify suspicious activities in crowded areas. HAR based on sensory data can be further divided into external sensors and body-worn sensors. When using external sensors, they are placed in the environment. IoT based systems such as in smart homes contain multiple sensors that monitor the activities and are used for home automation. Body-worn sensor based HAR systems have a major advantage over other methods as the activity monitoring zone is not limited to a confined region.

Among various applications of body-worn sensor HAR systems, below are few of them:

- **Healthcare and monitoring:** Due to the increase in life expectancy, considerable number of people will be over the age of 60 years in the next decades. To manage urgent medical situations, shorten the hospital stay and regular medical visits, HAR systems are employed. An HAR system for such an application, in addition to tracking activities, alert in case of emergency or if a fall is detected. Such systems also contain other auxiliary sub-systems for monitoring vital parameters (such as blood pressure,

heart rate, pulse and respiratory rate) and for connectivity over the internet (for transmission of such information).

- **Industrial applications:** In recent trends of Human-Robot Interaction (HRI), workers in manufacturing and assembly lines co-ordinate and are often assisted by industrial robots. These HRI systems use HAR subsystems, those assess the present state and activity of the human and are able to predict the future events involved in the workflow. Upon having this information, the HRI system instructs the robots to safely maneuver and co-ordinate with the operator to get the task done.

A specific case of activity recognition with body-worn sensors is the use of smartphones. Due to the availability of various embedded sensors and the improved ubiquitous capabilities, smartphones have become the best choice of daily-activity recognition applications.

For this project, data from the inertial sensors (accelerometer and gyroscope) is chosen. Accelerometer and gyroscope measure the linear acceleration and angular velocity of the smartphone. Although various additional sensors can be used together with the accelerometer and gyroscope, previous works like [17] and [18] have shown that additional sensors show very low improvements compared to using these sensors alone.

3.2 Datasets

3.2.1 Human Activities and Postural Transitions Dataset

The HAPT dataset is publically available at: https://archive.ics.uci.edu/ml/datasets/Smartphone-Based+Recognition+of+Human+Activities+and+Postural+Transitions.

The dataset contains data from the tri-axial accelerometer and gyroscope of a smartphone, both captured at a frequency of 50 Hz. The dataset consists of six basic activities (static and dynamic) and six postural transitions between the static activities. The static activities include standing, sitting and lying. The dynamic activities include walking, walking downstairs and walking upstairs. Stand-to-sit, sit-to-stand, sit-to-lie, lie-to-sit, stand-to-lie, and lie-to-stand are the classes for postural transitions between the static activities. Note that the postural transition classes have less samples than the basic activity classes, which results in an imbalanced dataset.

The dataset contains:

- **Raw signal data** of all the experiments from all participants and the labels of the performed activities
- **Records of activity windows** composed of feature vectors with time and frequency domain variables and their labels

We will use the raw data files in this project. The dataset has to be split further into train, validation, and test datasets based on the subjects. Use the following splits for the datasets from the raw dataset (the train/validation/test split is approximately 70 %/10 %/20 %):

- Train dataset: user-01 to user-21
- Validation dataset: user-28 to user-30

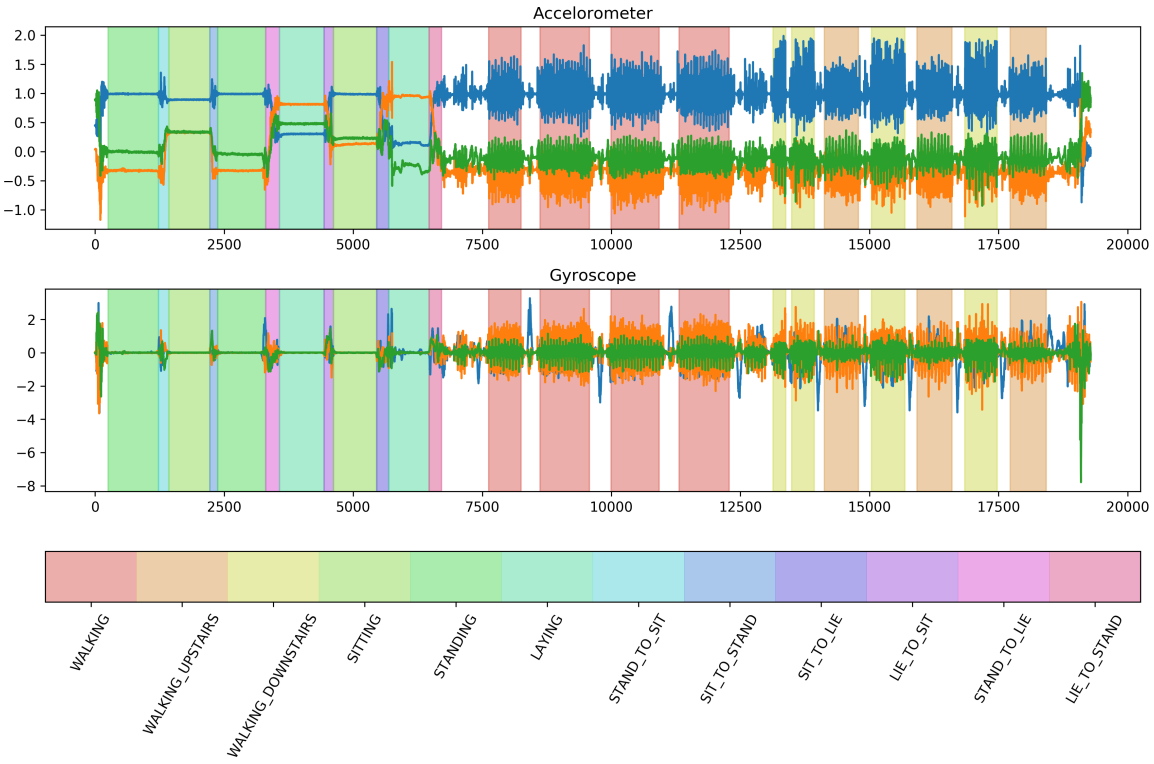

Figure 3.2: Example of the accelerometer and gyroscope signals and their labels

- Test dataset: user-22 to user-27

Number of activities	12
Number of subjects	30
Age group	19 to 48 years
Number of positions	1 (waist)
Number of sensors recorded	2
Available data formats	txt

Table 3.1: Details of the HAPT Dataset

Additional Hints

1. The units used for the accelerations (total and body) are 'g's (gravity of earth which is 9.80665 $\mathrm{m\,s^{-2}}$.
2. The gyroscope units are $\mathrm{rad\,s^{-1}}$.
3. White spaces in Figure 3.2 indicate unlabelled data. There are generally two ways how we can deal with unlabelled data in the case of time series. Either we sort out the data during our input pipeline, which results in a certain amount of data that is lost for training or we can ensure that the loss is masked out for unlabeled data. Latter results in a parameter update that is only based on labeled data. This can be achieved by

mapping these samples to an all-zero label in the case of using categorical crossentropy and providing a -1 label in the case of sparse categorical crossentropy.

3.2.2 [Optional] Real World (HAR) Dataset

The RealWorld (HAR) dataset is to be used for HAR with multiple positions. The dataset is available at https://sensor.informatik.uni-mannheim.de/#dataset_realworld. While most publicly available datasets were collected from an indoor/ laboratory environment, the activities of the RealWorld (HAR) dataset were carried out in a real world scenario.

The dataset contains signals from various smartphone embedded sensors like the accelerometer, Global-Positioning-System (GPS), gyroscope, light, magnetic field and sound. The activities that are covered by this dataset are climbing downstairs, climbing upstairs, jumping, lying, standing, sitting, running/jogging, and walking. Smartphones were placed at different body positions: chest, head, shin, thigh, upper arm, waist, and a smartwatch at the forearm that were recorded simultaneously. Figure 3.3 shows the placement of the sensors. Each activity is performed by each subject for around 10 minutes. Jumping activity was performed for 1.7 minutes due to physical exertion. Be aware that this difference leads to an imbalanced dataset.

It is also interesting to note how the subjects are distributed based on their fitness, gender and physical characteristics (Figure 3.4). [19] studies similarities between subjects with similar characteristics. So splitting subjects into train, validation, and test sets can be done based on Figure 3.4.

Data from subjects 4, 6, 7, and 14 can be dropped (do not consider them in any dataset) since their data is not consistent. With the remaining subjects, use the following splits for the datasets (the train/validation/test split is approximately 70 %/10 %/20 %):

- Train dataset: Subjects 1, 2, 5, 8, 11, 12, 13, and 15
- Validation dataset: Subject 3
- Test dataset: Subjects 9 and 10

Number of activities	8
Number of subjects	15
Age group	19 to 44 years
Number of positions	7
Number of sensors recorded	6
Available data formats	csv, sqlite

Table 3.2: Details of the RealWorld (HAR) Dataset

Note: Although 6 different sensors are available, only consider accelerometer and gyroscope data for this project. Since each sensor is tri-axial, the input to the model will consist of 6 (3 + 3) channels in total.

Figure 3.3: Sensor placement of the RealWorld (HAR) dataset [20]

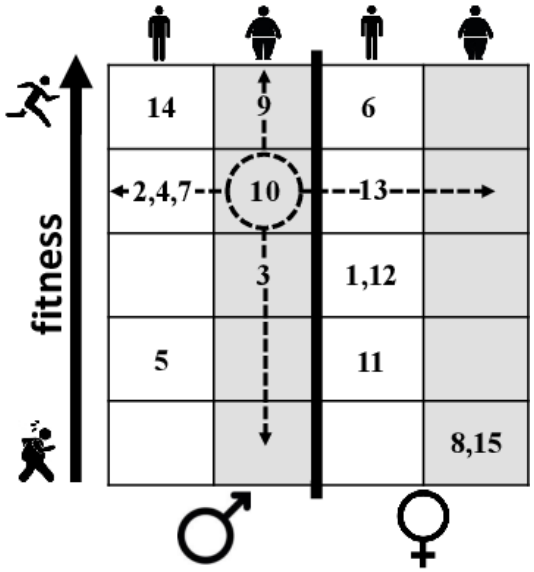

Figure 3.4: Distribution of users based on fitness, gender and physical characteristics [19]

Data preprocessing

The RealWorld (HAR) dataset is an example of data in a practical real-world situation. In practice, there are multiple preprocessing steps between data collection and the use of data. Some of the points here will give an idea about the steps. Consider the points below to ensure correctness in your preprocessing script. (Discuss with the tutors to know more about the possible issues to keep in mind when dealing with this dataset)

- Check for the availability of both accelerometer and gyroscope data.

- Check if the start time stamps of the sensors are aligned.
- Remove samples corresponding to 5 seconds from the beginning and end of the file, as they are only adding “noise“.
- Check for incorrect alignment between the sensors. (In some cases, it is possible that the samples and their time stamps of the two sensors are not synchronized. This can be a problem when channels from two sensors are concatenated.)

3.3 Tasks

The task in this project is to predict human activities from inertial sensor data using (recurrent) neural networks. Similar to the previous project, build a generic framework following the steps in the subsequent sections.

3.3.1 [Optional] Record your own data (Android only)

It is also possible to collect and test your own data with your smartphone. You can use the Android App “Sensors Data Collector“ available on the Google Play Store. Use the application to collect data in the RealWorld (HAR) Dataset format.

The app allows you to access a variety of sensors in your smartphone. After selecting the sensors to be used, the sensor data can be labeled, recorded and saved in a csv file. It is recommended to record and save the individual activities in separate datasets. This simplifies the subsequent processing of the data.

For the data preprocessing the same notes apply as in the previous section 3.2.2.

Since the collection of an HAR dataset is very time-consuming, it is possible to use the RealWorld (HAR) dataset as the training dataset and to test the pre-trained model with your own data. The prerequisite for this is that the data is collected in a comparable way. Please refer to the information from https://sensor.informatik.uni-mannheim.de/#dataset_realworld.

Below are some hints to avoid noisy data during collection:

- Keep the orientation of the smartphone intact during data recording.
- Use a strap or smartphone holder (as in Figure 3.3) to avoid orientation errors between users.

3.3.2 Input pipeline

Task: Use the ideas and experiences from the previous project for the input pipeline. Use `tf.data` to build the data input pipeline. Note that the end of the pipeline or the input to the model will be a 6-channel input (3 accelerometer and 3 gyroscope axes).

Input normalization:

Large offsets and strongly differing variances in the input data affect the training performance. When dealing with multi-channel (multi-variate) time series data, normalized data produces better performance. Z-Score normalization is the most commonly used normalization technique resulting in zero mean and unit variance. Normalization has to be performed for each channel independently. In this case, normalization also removes user dependent biases and offsets due to device placement errors.

Sliding Window

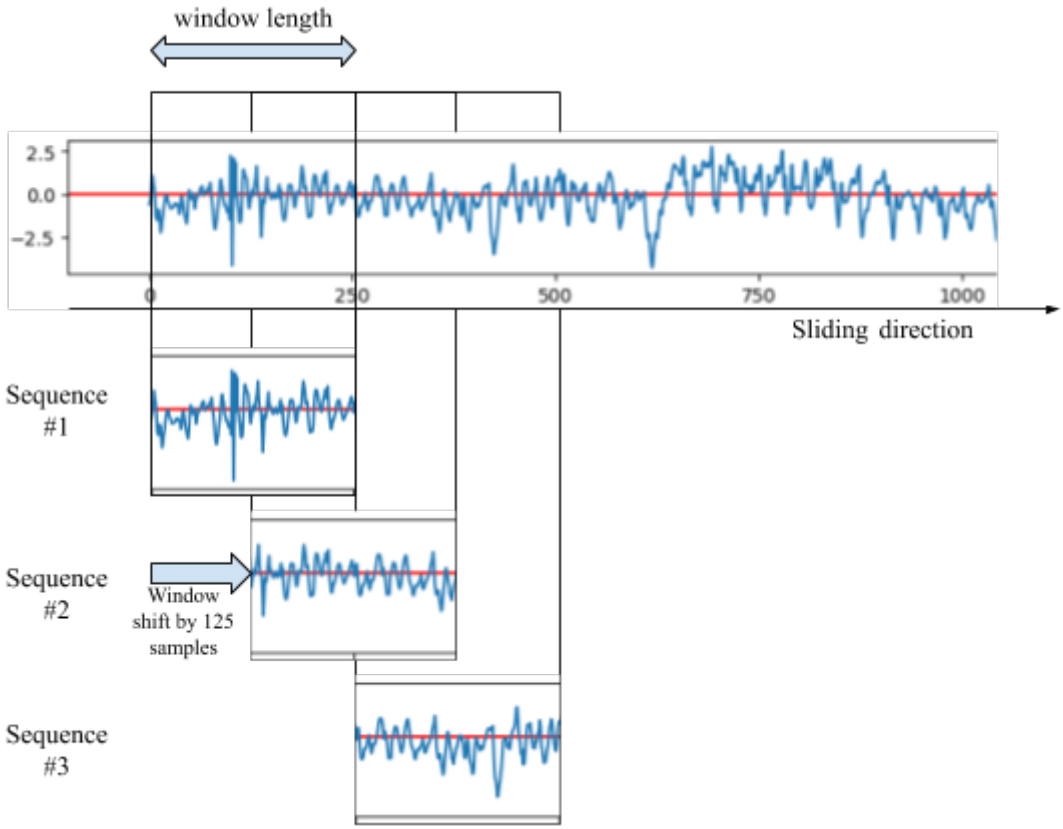

Figure 3.5: Example of use of sliding windows to create fixed-length sequences

The data after preprocessing will contain long sequences of varying lengths. In order to have fixed-length input sequences, we employ the **sliding window** technique. Note that this is also a data augmentation step for time-series data. There are two parameters to be considered here, namely:

- **Window length:** The longer the window length, the longer is the information content. The model is then fed with more information per sequence. For a given sampling rate, the window length is a hyper-parameter for time-series classification.

  For this step, consider a window size of 250 samples (that corresponds to 5 seconds of data at 50 Hz sampling rate of the sensors). Keep in mind not to hard-code the value.

Consider it configurable so that it can be tuned during the next (optional) hyperparameter optimization step.

- **Window shift:** It defines the amount of shift (or slide) of the sliding window. Consider shifts by 125 samples (with 50 % overlap). This parameter is also a hyperparameter that can be considered for the optimization step.

Consider using the batch function (https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset#batch) or the window function (https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/data/Dataset#window) to create batches/windows and shifts. The `drop_remainder` argument comes handy to eliminate samples at the end that do not fit the window size.

After the initial correctness checks and normalization, you can store the extracted windows as csv or TFRecords (recommended). To learn more about TFRecords, check https://www.tensorflow.org/tutorials/load_data/tfrecord.

3.3.3 Model architecture

Task: Similar to the previous project, you will choose your own architecture for the activity recognition tasks. Again, use the ideas from Section 2.3.2 for your model. Start with a simple RNN model for the task with a single LSTM layer. An LSTM layer is able to learn strong temporal dependencies between the samples of the given data. Optionally, you can also consider using other models with Vanilla RNN or GRU layers.

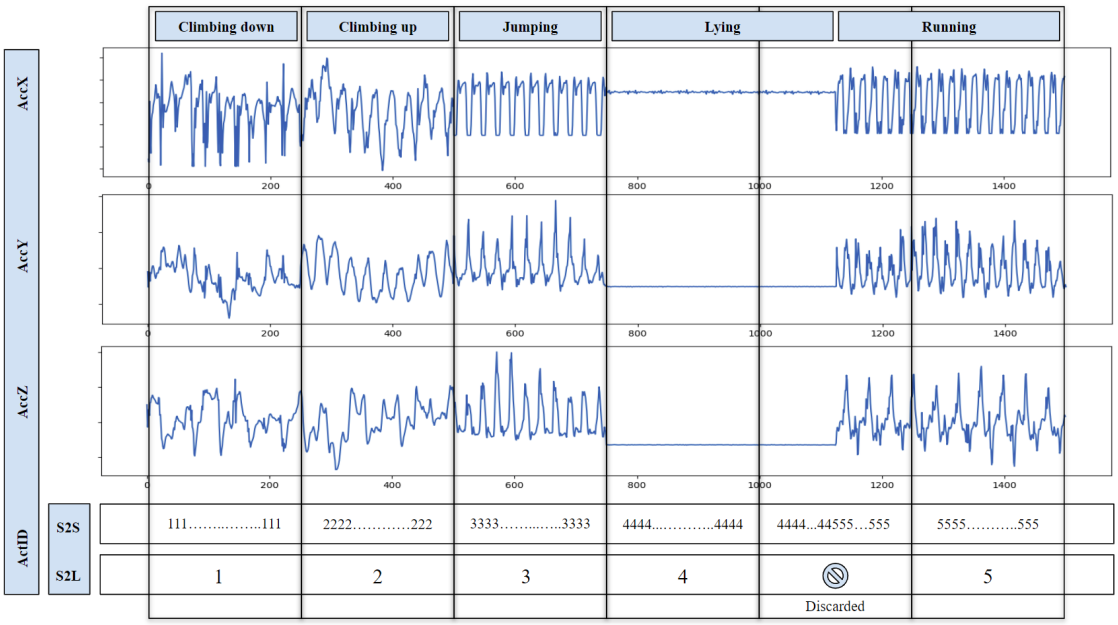

Figure 3.6: Example of S2S and S2L classification tasks

Depending on the type of output labels, there are two types of classification tasks possible.

- Sequence-to-Sequence (S2S): The model for this classification task consists of an LSTM last layer that returns sequences as the output. For an input sequence with T samples, the output sequence also consists of a sequence of T samples.

- Sequence-to-Label (S2L): The last LSTM layer returns a single value (value at the last time step) as the output. Thus the model takes an input sequence with T samples and outputs a single label. Keep in mind that all the samples in a given input sequence should belong to the single class, whose label will be used as the ground truth to calculate the loss.

3.3.4 Activity recognition for a single position

Task: Use the HAPT dataset, which was recorded with the smartphone positioned at the waist, to predict the activities performed by humans. Visualize the result for a whole sequence from the test set.

Optionally, you can use the RealWorld (HAR) dataset as well. It contains data for muliple positions, just pick a single one to perform the activity recognition. To get an idea about the performance for different positions, have a look at [21] (feature extraction based).

3.3.5 Metrics

To evaluate the performance of the model, you can use the same metrics such as confusion matrix and (balanced) accuracy, as in the first project. In addition, for HAR classification, F-measure, recall, and precision are commonly used.

3.3.6 [Optional] Hyperparameter Optimization

Start with a simple model with a single LSTM layer and then try to improve the model by tuning the hype-parameters. For time-series tasks, the window size and the window shift are important parameters that affect the performance. To simplify this task, we recommended a window size and window shift of 250 and 125 samples (50 % overlap), respectively. These parameters are specific to inertial sensors that are sampled at 50 Hz frequency only. If you want to extend the project to other sensors, you should consider different sizes and shifts as hyperparameters as well. You can also consider these parameters to be hyperparameters to see how the performance varies.

For an LSTM based model (for any RNN Model in general), some of the model hyperparameters are:

- Number of recurrent layers
- Number of fully-connected layers
- Number of hidden units
- Dropout rate
- Optimizer, learning rate, decay, ...
- Statefulness (stateful/stateless)
- ...

Grid Search

The simplest way to ensure coverage of all hyperparameter combinations is grid search, which is simply an exhaustive search method. Hyperparameters to be considered for the search are manually specified in advance. Every combination of the parameters is then fed to the model for training.

You can use TensorBoard to visualize the performance for different hyperparameters using the TensorBoard HParams plugin (https://www.tensorflow.org/tensorboard/hyperparameter_tuning_with_hparams).

Neural networks are often much more sensitive to some parameters than others, so random search will often give you a similar result, requiring much less time.

Bayesian Optimization

A more sophisticated way of performing hyperparameter optimization is Bayesian optimization. A Bayesian optimization problem tries to find the minimum/maximum of a function $f(x)$ on some bounded set χ, which is a subset of $\mathbb{R}^D$. Bayesian optimization constructs a probabilistic model for $f(x)$ and then exploits this model to make decisions about where in χ to evaluate the function, while integrating out uncertainty. This technique not only relies on the local gradient and Hessian approximations, but on all the information available from the previous evaluations of $f(x)$. Eventually, a minimum/maximum of a difficult non-convex function can be obtained with relatively few evaluations compared to other optimization methods. [22]

A python implementation of the Bayesian optimization is available as a package under the MIT License and can be installed using pip or conda. Refer to https://github.com/fmfn/BayesianOptimization for more details on Bayesian optimization and tutorials on how to use it.

In most cases, you might want to generate discrete function parameters. For the given bound, the generated parameters are real. The package does not support working with discrete parameters. To work with discrete parameters, check the hints provided in the advanced-tour (https://github.com/fmfn/BayesianOptimization/blob/master/examples/advanced-tour.ipynb) section. Or as an alternative, you can round the real value to the nearest integer value.

3.3.7 [Optional] Activity recognition using multiple positions

The RealWorld (HAR) Dataset is available with data from multiple positions. In this task, the idea is to use the sensors from multiple positions to perform the activity recognition. Compare the final results to the results from individual positions.

3.3.8 [Optional] Deployment on an Android smartphone

TensorFlow Lite provides APIs to convert TensorFlow models to run on mobile and embedded platforms. Now, use this to convert the model to a `.tflite` file that can be used within

an Android application.

Note: TensorFlow 2.0 has limited support to serialize and hence convert RNN models to the `.tflite` file format. Check the latest GitHub issues of TensorFlow to track the progress. TFLite is only able to convert specific operations for now, see https://www.tensorflow.org/lite/convert/rnn.

You can experiment with a non-RNN model such as a model with a 1D Conv layer. You can find some examples to convert your model to a `.tflite` file here: https://www.tensorflow.org/lite/convert/python_api.

Android Studio

Use the hints below to work with Android Studio for the deployment app.

- Create a blank project
- Place the `.tflite` file in the assets folder
- Register the sensors (accelerometer and gyroscope) under sensor manager
- Use the same sampling rate as that used for data collection
- `OnSensorChanged` event, store the sensor values into a data buffer (for example, a list with size 250 samples)
- Implement a method that calls the tflite model using `TensorflowInferenceInterface`
- When the data buffer is full, call the method to predict probabilities
- Finally use these probabilities to get the output class label

References for Android studio: https://developer.android.com/ml.
TensorFlow Lite examples: https://github.com/tensorflow/examples/tree/master/lite/examples

Bibliography

[1] Porwal, Prasanna ; Pachade, Samiksha ; Kamble, Ravi ; Kokare, Manesh ; Deshmukh, Girish ; Sahasrabuddhe, Vivek ; Meriaudeau, Fabrice: Indian diabetic retinopathy image dataset (IDRiD): a database for diabetic retinopathy screening research. In: *Data* 3 (2018), Nr. 3, S. 25

[2] Goh, James Kang H. ; Cheung, Carol Y. ; Sim, Shaun S. ; Tan, Pok C. ; Tan, Gavin Siew W. ; Wong, Tien Y.: Retinal imaging techniques for diabetic retinopathy screening. In: *Journal of diabetes science and technology* 10 (2016), Nr. 2, S. 282–294

[3] Huang, Jonathan ; Rathod, Vivek ; Sun, Chen ; Zhu, Menglong ; Korattikara, Anoop ; Fathi, Alireza ; Fischer, Ian ; Wojna, Zbigniew ; Song, Yang ; Guadarrama, Sergio u. a.: Speed/accuracy trade-offs for modern convolutional object detectors. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, S. 7310–7311

[4] Selvaraju, Ramprasaath R. ; Cogswell, Michael ; Das, Abhishek ; Vedantam, Ramakrishna ; Parikh, Devi ; Batra, Dhruv: Grad-cam: Visual explanations from deep networks via gradient-based localization. In: *Proceedings of the IEEE International Conference on Computer Vision*, 2017, S. 618–626

[5] Silver, David ; Huang, Aja ; Maddison, Chris J. ; Guez, Arthur ; Sifre, Laurent ; Van Den Driessche, George ; Schrittwieser, Julian ; Antonoglou, Ioannis ; Panneershelvam, Veda ; Lanctot, Marc u. a.: Mastering the game of Go with deep neural networks and tree search. In: *nature* 529 (2016), Nr. 7587, S. 484

[6] Krizhevsky, Alex ; Sutskever, Ilya ; Hinton, Geoffrey E.: Imagenet classification with deep convolutional neural networks. In: *Advances in neural information processing systems*, 2012, S. 1097–1105

[7] Maaten, Laurens van d. ; Hinton, Geoffrey: Visualizing data using t-SNE. In: *Journal of machine learning research* 9 (2008), Nr. Nov, S. 2579–2605

[8] McInnes, Leland ; Healy, John ; Melville, James: Umap: Uniform manifold approximation and projection for dimension reduction. In: *arXiv preprint arXiv:1802.03426* (2018)

[9] Springenberg, Jost T. ; Dosovitskiy, Alexey ; Brox, Thomas ; Riedmiller, Martin: Striving for simplicity: The all convolutional net. In: *arXiv preprint arXiv:1412.6806* (2014)

[10] Sundararajan, Mukund ; Taly, Ankur ; Yan, Qiqi: Axiomatic attribution for deep networks. In: *Proceedings of the 34th International Conference on Machine Learning-Volume 70* JMLR. org, 2017, S. 3319–3328

[11] Donahue, Jeff ; Jia, Yangqing ; Vinyals, Oriol ; Hoffman, Judy ; Zhang, Ning ; Tzeng, Eric ; Darrell, Trevor: DeCAF: A Deep Convolutional Activation Feature for Generic Visual Recognition. In: *CoRR* abs/1310.1531 (2013)

[12] Oquab, M. ; Bottou, L. ; Laptev, I. ; Sivic, J.: Learning and Transferring Mid-level Image Representations Using Convolutional Neural Networks. In: *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, S. 1717–1724

[13] Tajbakhsh, N. ; Shin, J. Y. ; Gurudu, S. R. ; Hurst, R. T. ; Kendall, C. B. ; Gotway, M. B. ; Liang, J.: Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning? In: *IEEE Transactions on Medical Imaging* 35 (2016), May, Nr. 5, S. 1299–1312

[14] Yosinski, Jason ; Clune, Jeff ; Bengio, Yoshua ; Lipson, Hod: How transferable are features in deep neural networks? In: *Advances in neural information processing systems*, 2014, S. 3320–3328

[15] Raghu, Maithra ; Zhang, Chiyuan ; Kleinberg, Jon ; Bengio, Samy: Transfusion: Understanding transfer learning for medical imaging. In: *Advances in Neural Information Processing Systems*, 2019, S. 3342–3352

[16] Goodfellow, Ian ; Bengio, Yoshua ; Courville, Aaron: *Deep Learning*. MIT Press, 2016. – http://www.deeplearningbook.org

[17] Zhang, Mi ; Sawchuk, Alexander: A Feature Selection-based Framework for Human Activity Recognition Using Wearable Multimodal Sensors. In: *Proceedings of the 6th International Conference on Body Area Networks*. ICST, Brussels, Belgium, Belgium : ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2011 (BodyNets '11). – ISBN 978–1–936968–29–9, 92–98

[18] souza, Warren ; Kavitha, R.: Human Activity Recognition Using Accelerometer and Gyroscope Sensors. In: *International Journal of Engineering and Technology* 9 (2017), 04, S. 1171–1179. http://dx.doi.org/10.21817/ijet/2017/v9i2/170902134. – DOI 10.21817/ijet/2017/v9i2/170902134

[19] Sztyler, Timo ; Stuckenschmidt, Heiner: Online personalization of cross-subjects based activity recognition models on wearable devices. In: *2017 IEEE International Conference on Pervasive Computing and Communications (PerCom)* (2017), S. 180–189

[20] Sztyler, Timo ; Stuckenschmidt, Heiner: On-body Localization of Wearable Devices: An Investigation of Position-Aware Activity Recognition. In: *2016 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, IEEE Computer Society, 2016, 1–9

[21] Sztyler, Timo ; Stuckenschmidt, Heiner ; Petrich, Wolfgang: Position-Aware Activity Recognition with Wearable Devices. In: *Pervasive and Mobile Computing* 38 (2017), Nr. Part 2, 281–295. http://dx.doi.org/10.1016/j.pmcj.2017.01.008. – DOI 10.1016/j.pmcj.2017.01.008

[22] Snoek, Jasper ; Larochelle, Hugo ; Adams, Ryan P.: Practical Bayesian Optimization of Machine Learning Algorithms. Version: 2012. http://papers.nips.cc/paper/4522-practical-bayesian-optimization-of-machine-learning-algorithms.pdf. In: Pereira, F. (Hrsg.) ; Burges, C.

J. C. (Hrsg.) ; Bottou, L. (Hrsg.) ; Weinberger, K. Q. (Hrsg.): *Advances in Neural Information Processing Systems 25*. Curran Associates, Inc., 2012, 2951–2959