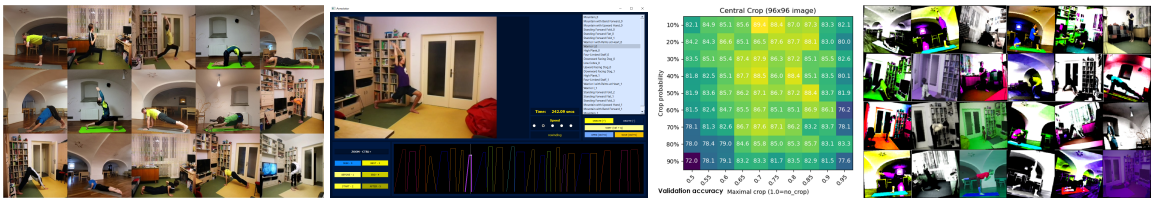
BRNO UNIVERSITY OF TECHNOLOGY
FACULTY OF INFORMATION TECHNOLOGY

Excel@FIT 2021
http://excel.fit.vutbr.cz

Detection of Yoga Poses in Image and Video

Jiří Kutálek



Abstract

In this paper, the concept of a smartphone app detecting Yoga poses and displaying several frames to a user is presented. The goal of this project is proving that even a simple Convolutional Neural Network (CNN) model can be trained to recognize and classify video frames from a Yoga session. I created an application in which the videos are manually annotated. The data, consisting of frames captured from 162 collected videos based on the annotations, is then passed to train a CNN model. The Dataset consists of 22 000 images of 22 different Yoga poses. The frames are captured using the OpenCV library, the training process is handled by the TensorFlow platform and the Keras API, and the results are visualized in the TensorBoard toolkit. The Model's multi-class classification accuracy reaches 91% when the binary cross-entropy loss function and the sigmoid activation function are used. Despite the experimental results are promising, the main contributions are the dataset forming tools and the Dataset itself, which both helped to confirm the proof-of-concept.

Keywords: Yoga Poses Detection — Video Annotation Application — Training CNN for Yoga Poses Recognition

Supplementary Material: Demonstration Video

*xkutal09@stud.fit.vutbr.cz, *Faculty of Information Technology, Brno University of Technology*

1. Introduction

Over the last few years, Yoga popularity is rapidly increasing. Due to this, there are plenty of instructional videos and Yoga teaching smartphone applications available on the internet. They usually provide a decent workout guide including useful information for practicing Yoga on one's own. However, they hardly ever provide a visual workout feedback. This can be obtained either by exercising in front of the mirror, or by recording the session on camera and exploring the video. Both might be quite inconvenient.

This paper introduces an idea of a smartphone app providing the workout feedback easily, specifically by choosing and showing just a few frames, representing the performed Yoga poses, from the recorded workout session. Thanks to that, a user does not need to manually seek frames in the video. The main problem obviously is, how does the application know, which of the many frames in the video to choose and display to the user? I design a Convolutional Neural Network (CNN) classifying the Yoga images into categories based on the corresponding Yoga poses.

In 2016, Convolutional Pose Machines (CPMs) [1] presented an innovative systematic design for how convolutional networks can be incorporated into the pose machine framework [2] for learning image features and image-dependent spatial models for the task of pose estimation. CPM proposed a sequential architecture composed of CNNs that directly operate on belief maps from previous stages, producing increasingly refined estimates for part locations, without the need for explicit graphical model-style inference.

An approach to accurately recognize various Yoga poses using deep learning algorithms [3] has been presented in 2019. A hybrid deep learning model is proposed using CNN and LSTM [4] for Yoga recognition on real-time videos, where CNN layer is used to extract features from keypoints of each frame obtained from OpenPose [5] and is followed by LSTM to give temporal predictions.

Unfortunately, these methods are not suitable for a smartphone app as the architecture is too heavy for the processing units currently in use. Besides that, another paper [6] proposes a system monitoring body parts movement and accuracy of different Yoga poses, which aids the user to practice Yoga. However, it uses Microsoft Kinect for human body parts real time detection, an expensive device most people do not have at hand.

This paper proposes a more practical solution. All the user has to do is to record their workout session on a camera and then run the application. After picking the input video, several well performed Yoga images are selected and displayed on the screen. The great benefit is that reviewing a dozen photos is much more accessible than watching and rewinding a whole session video. Furthermore, a person is able to either evaluate images by their eyes, or save them to gallery and consult with their Yoga instructor later, which may both lead to the exercise progression.

Up till now, I collected many Yoga videos serving as data resources and created a custom annotation application enabling to form the Dataset from them. For this purpose, I wrote a script capturing video frames by the annotations, which forms the Dataset, currently containing tens of thousands of images. Apart form that, I proposed a CNN model and performed many experiments in order to tune it as well as possible.

2. Background

Convolutional Neural Networks are used for solving the image classification problems because of their high accuracy. The principles behind them, including an insight into their architecture, are described in this Section. On top of that, the tools utilized in this work are presented at the end.

2.1 Brief Introduction to the CNNs

Using an ordinary Artificial Neural Network (ANN), image classification problems become difficult because 2D images need to be converted to one-dimensional vectors. This increases the number of trainable parameters rapidly, which takes storage and processing capability. Convolutional Neural Network, a class of neural networks, convolves the learned features with input data, and uses 2D convolutional layers, making this architecture well suited to processing 2D data, such as images.

A CNN typically consists of several convolutional and pooling layers dealing with feature extraction followed by fully connected layers managing the classification itself (Figure 1). An activation function plays an important role in feature extraction allowing to classify even non-linearly separable data. The complexity of the features detected typically grows with the layer depth.

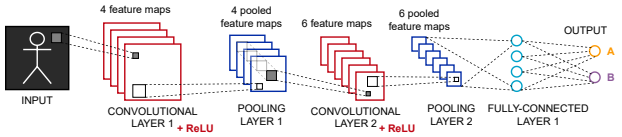


Figure 1. Schematic of a Convolutional Neural Network. The feature maps typically gradually decrease their spatial resolution, while increasing the number of channels. Ultimately, the CNN extracts more and more abstract information and produces a decision with typically very small number of outputs.

Convolution is performed on an input image using a filter or a kernel (small matrix of values), sliding over the image, multiplying its values with the image pixel values and adding them up. The output values are passed through an activation function, adding non-linearity to a network. A commonly used ReLU function zeros the negative values and keeps the positive ones. The convolutional layer outputs are usually referred to as feature maps or channels. Pooling (subsampling) layer then decreases the feature map size to reduce the number of computational parameters in the network. For instance, max pooling, a frequently used type of pooling, takes the maximum value in a specified window. After a few combinations of convolutional and pooling layers, the final output is flattened and fed into a fully connected layer (a regular ANN) for classification purposes.

In 2012, the AlexNet CNN architecture [7] (Fig. 2) dominated the ImageNet ILSVRC challenge [8] and started a wave of interest in CNNs as one of the first models stacking convolutional layers directly on top of each other without inserting pooling layers between them. Moreover, it implements the **Dropout** regularization method [9] to reduce overfitting in the fully connected layers.

CNNs were starting to get deeper, and the simplest way of improving the deep networks performance is by increasing their size. Visual Geometry Group (VGG) presented the VGG-16 [10], consisting of 16 weighted layers and 136 M parameters in total.

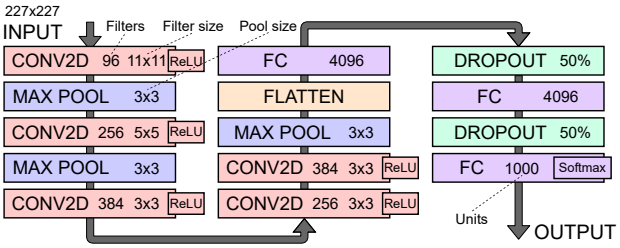


Figure 2. Schematic of the AlexNet Convolutional Neural Network architecture. The model contains five convolutional layers, whose output is passed through the *ReLU* activation function, combined with three max pooling layers together taking care of feature extraction. During this process, the image resolution is reduced from 227×227 px to 6×6 px. Then the output is flattened and passed to the second part of the network managing the classification by three fully connected layers, coupled with two Dropout layers randomly dropping 50 % of outputs each. The *softmax* activation function is used at the output layer to make predictions. The network operates with more than a 56 M trainable parameters.

In 2015, Microsoft Research proposed another successful model, an extremely deep network called ResNet [11] composed of 152 layers, known for the introduction of residual blocks.

I made few experiments with the AlexNet and VGG-16 CNN models. The results are presented in Section 5.

2.2 Work Tools

I capture video screenshots through the OpenCV library [12, 13] and form a dataset from them.

The training process itself is in a full control of the TensorFlow platform [14, 15] (v2.3 and higher) and its associated deep learning API Keras [16]. I use Keras for dataset loading and configuration, as well as the CNN model definition and its subsequent training. Thanks to the TensorFlow's `tf.summary` module I am able to write summary data and visualize them through the TensorBoard toolkit[1] (Figure 3).

All of the scripts are written in the Python[2] programming language.

3. Forming the Dataset

With the help of my supervisor, I collected 162 Yoga videos containing three different Yoga sequences and build a dataset from them. Unfortunately, videos currently come only from two people, which is a too small sample size to form a truly comprehensive dataset. But for now the goal is the proof-of-concept (verifying the potential of the idea) and that purpose the Dataset serves well.

[1] https://github.com/tensorflow/tensorboard

[2] https://docs.python.org/3/reference/

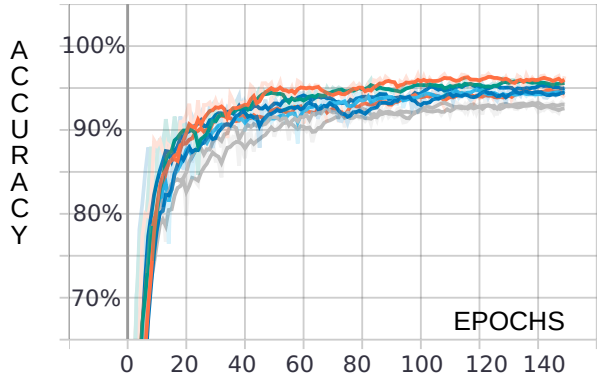


Figure 3. Example of the training process visualization in TensorBoard. This diagram shows eight validation accuracy curves, each representing a single model training process. The x-axis represents the training epochs, while the y-axis shows the accuracy. The TensorBoard's Scalars Dashboard allows a user to visualize many different accuracy or loss curves at once, filter out the unwanted ones, compare the wanted ones and find out additional information about each epoch, such as its duration or the actual accuracy value. Moreover, the curves are updated step by step during the training process, making it possible to monitor the training process and react to changes.

3.1 Annotating Videos using a Custom Annotation Application

The Dataset for the training itself consists of images, not videos. Therefore, I created an annotation tool (Fig. 4), where the videos are manually annotated. The annotations define exact frames, at which the training and validation frames are being taken.

In the video which the application takes on the input, a person performs one of the predefined Yoga sequences, each of them containing several Yoga poses (Figure 5). And each of the poses exercised in the particular sequence is represented by an annotation. This means that when a pose is done three times during its corresponding sequence, there is a single annotation for each of them.

The annotations (Figure 6) consist of a pose identifier and five numerical values representing the video frames defining the time a person is performing the corresponding Yoga pose. In the app (Figure 4), the frames are marked by the buttons in the bottom left corner and the annotations are then displayed in the video timeline.

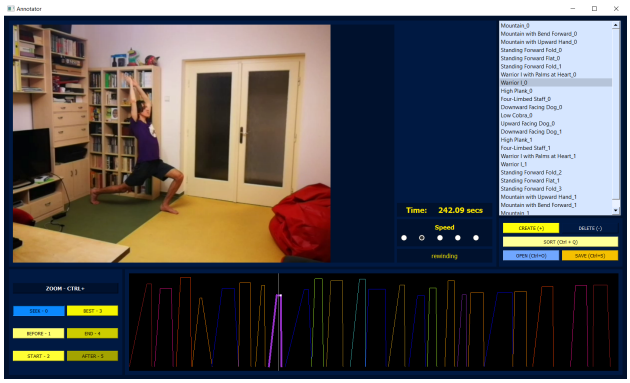


Figure 4. Screenshot of the annotation application workplace. In the top right corner there is a list of annotations belonging to the currently processed video. The annotations are created by the yellow buttons at the bottom left and visualized by graphs in the timeline. In addition, the application supports video rewinding and speed changes and provides several other features such as video seeking and timeline zooming.

Once done and the progression is saved, a `.json` file, named by the processed video file, is created. There is a single file for each annotated video storing all of the annotations related. The `.json` files storing the annotations are passed to a script randomly taking screenshots at the specified frames using the OpenCV library, which forms the Dataset.

Figure 5. Examples of Yoga poses representing the classes into which the data is classified.

3.2 Data Shape and Volume

Despite the 162 videos come from only two subjects and the fact that each subject always exercises in a same room, each of the videos is unique. Most of the Yoga sessions are recorded by multiple cameras (up to four) positioned in various angles. Furthermore, almost for every session the subjects wear clothes of different colors and usually change the part of a room where practicing, so that the video background is altered. The light conditions for some sessions vary too, but care should be taken to ensure enough light and not much shadow in a footage. Thanks to all these aspects

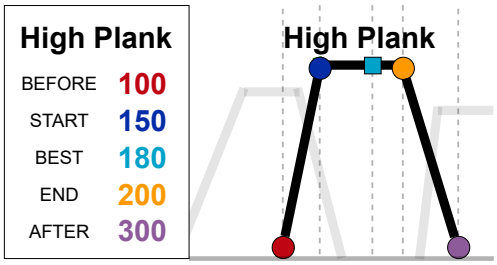


Figure 6. Example of an annotation referring to the **High Plank** Yoga pose. The graph on the right portrays its corresponding representation in the timeline. All of the frames between *START* and *END* symbolize the time a person is performing the pose, and therefore, the screenshots can be taken during this period. The frame intervals between *BEFORE* and *START* and between *END* and *AFTER*, as well as the *BEST* frame in each annotation, are not utilized for now, but could be useful in a future work.

the data is quite varied, although the number of people producing the Yoga videos is very low.

The Yoga videos are split into two directories separating the training and the validation data by $3:1$. This is done at this early level, because of the potential to manually choose the source of pictures used for model training (I am able to build both easy and hard datasets). The frame extraction script walks through the directory containing the video files iteratively, meaning that all of the classes hold the same amount of data.

Currently, I work with the Dataset (Figure 7) containing 44 000 images (2 000 per class/Yoga pose), but thanks to the frame range defined in the annotations, I am able to create a dataset carrying hundreds of thousands of pictures. The frames are captured with the fixed resolution 256×256 px (squared) and resized at a later stage, when the Dataset is being configured for performance, into an appropriate shape.

However, not all of the 44 000 images are used further in the training process. The frame extraction script serves only as a giant image collection provider and this collection is updated only when new videos are being integrated into the dataset forming process, which is done occasionally. Moreover, I often alter the Dataset size and structure and it takes more than an hour to walk through the videos and capture frames.

Therefore, I truly work with 22 000 images randomly chosen from the 44 000 before an every set of experiments. As mentioned above, the training and validation data is split by $3:1$, which means that the training dataset contains 16 500 images (750 per pose) and the validation dataset consists of 5 500 images (250 per pose).

Figure 7. Samples from the Dataset. The videos are square (or trimmed to square if captured otherwise). The subject should be visible during the whole training. A defined set of Yoga sequences was performed when shooting the videos.

4. Training CNN for Yoga Poses Recognition

A complete workflow of how the Dataset is loaded and configured, as well as how the actual CNN training process looks like, is described in this Section. As the training process is in a full control of the TensorFlow platform and the Keras API, a few examples of functions and utilities used are covered too.

4.1 Dataset Loading, Configuration and Arrangement

In preparation for model training, the images are loaded off disk using the *image_dataset_from_directory* utility, provided by the `tf.keras.preprocessing` module. The Dataset is already divided into training data and validation data, as the videos come from two different directories. The frame resolution remains at the 256×256 px and the image batch size is usually set to 32.

After loading, the data is normalized to unify the data distribution of the pixels (the input in the $[0, 255]$ range is rescaled to fit the $[0, 1]$ range) and then resized, mostly to 96×96 px, which is the image resolution I mainly operate with.

To fight overfitting in the training process and increase the diversity of the Dataset, I apply various augmentation techniques to the images (Figure 8). These transformations cover picture rotating, horizontal flipping and several image color enhancements, namely contrast, brightness, saturation and hue changes. The central region of some pictures is cropped, too. Most of the techniques are realized using the `tf.image` module. For frame rotations I utilize the `tfa.image` one belonging to the *TensorFlow Addons* repository.

All of these techniques are combined together and applied to each image batch of the training data before the first training epoch starts (for each epoch there are different transformations applied to the training data). Lastly, the prefetch transformation creates an overlap between the data being pre-processed and the model execution while training, allowing the later processed data to be prepared, while some other data is being processed.

Figure 8. Augmented image samples. Each of the pictures is taken from a different batch, as all the transformations applied are same for the images in a single batch. The augmentation techniques include color changes, rotations, flipping, and a central crop.

During the first training epoch, the loaded pictures are kept in memory (cached) for the subsequent iterations to use them, in order to prevent the Dataset becoming a bottleneck during the training process.

4.2 CNN Model Training Process

Before the actual training process starts, it is necessary to build the CNN Model. I do this through the `tf.keras.Sequential` class grouping a linear stack of layers into a single model. The models I experiment with consist of six to ten convolutional layers combined with four or five max pooling layers, followed by no more than three fully connected layers (see Figure 9 for an example). I try to design the architectures as simple as possible (although the highest priority is the model efficiency of course). As for the activation functions, I majorly use the Rectified Linear Unit (ReLU) at the hidden layers and switch between softmax and sigmoid at the output layer.

For model evaluation I use the *categorical accuracy* metric calculating how often predictions match one-hot labels, as the data is represented as one-hot Tensors (in one-hot encoding, each class is represented by a binary feature – either 1 or 0). Since the labels are provided in a one-hot representation, I use the *categorical cross-entropy* loss function to compute the cross-entropy loss between labels and predictions. However, sometimes I replace it with the *binary cross-entropy*

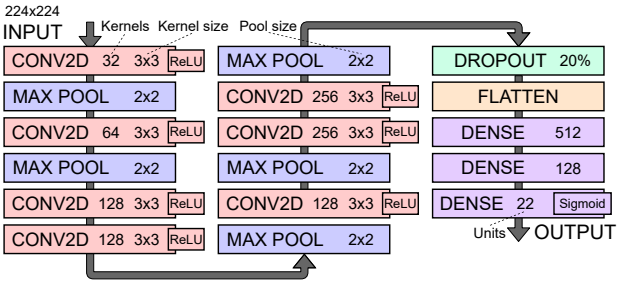


Figure 9. Schematic presenting one of the most successful CNN models I found during the experiments. It consists of eight convolutional layers, whose output is passed through the ReLU activation function, managing the feature extraction together with five max pooling layers, which consecutively reduce the image dimensionality from 224×224 px to 7×7 px. Before the final output is flattened and fed to the first fully connected layer, 20 % of the outputs are dropped (chosen randomly), in order to reduce overfitting. The Model contains three fully connected layers (called **Dense** in the Keras Sequential API). The sigmoid activation function is used at the output layer to make predictions. This Model operates with 7,7 M trainable parameters. Its accuracy is shown in Fig. 14.

loss function combined with the sigmoid activation function at the Model output layer (see Section 5 for more details). On top of that, I use Adam [17] as an optimizer.

The training process happens in epochs. The batches of training data are fed to the network one-by-one, followed by the validation data batches every epoch. For each batch, its samples are used to estimate the error gradient, which is subsequently used to update the model weights. At the end of each epoch, the learning rate, specifying how much are the model weights being updated, is recalculated.

Moreover, I use to visualize images from preferred batches, in order to both explore the augmented training data and see the correct and wrong label predictions on validation data. During each epoch, right after the Model weights are updated for the last time, scalar Tensor values (accuracy and loss) are written to disk using the `tf.summary` file writer. By doing so, I am able to visualize these metrics through TensorBoard and track the model training effectively.

5. Experimental Results

I experimented with CNN model architectures (including the AlexNet and VGG-16), data augmentation techniques and various training parameters such as learning rate, loss functions or setting the input image dimensions. Some of them led to interesting findings or helped to reach great results. In this Section, the experiments made, as well as the results achieved are presented.

5.1 Findings, Advancements and Model Tuning

Activation Functions and Loss Functions Perhaps the most important finding I have made is related to activation and loss functions. Since I do the multi-class classification, I used to work with the *softmax* activation function, normalizing the network output to a probability distribution over predicted classes, together with the *categorical cross-entropy* loss function. But, after a consultation with my supervisor, I found out that a combination of the *sigmoid* activation function and the *binary cross-entropy* loss function is more efficient.

This finding was confirmed across many CNN models, where all the networks implementing the sigmoid function had about 4 % higher validation accuracy, than the models using the softmax. From that moment, most of the models tested use the sigmoid activation function at the output layer instead of softmax.

Model Architectures I experimented with more than a hundred CNN models so far. Firstly, I tested the part of a CNN model dealing with feature extraction (convolutional and max pooling layers). Working with the 96×96 px image resolution, I found out that the Model should contain at least four consecutive combinations of convolutional and pooling layers to make decent predictions. Besides, I ascertained that a too low number of convolutional filters leads to worse validation accuracy and more unstable results (an example of this kind of model is shown in Fig. 10).

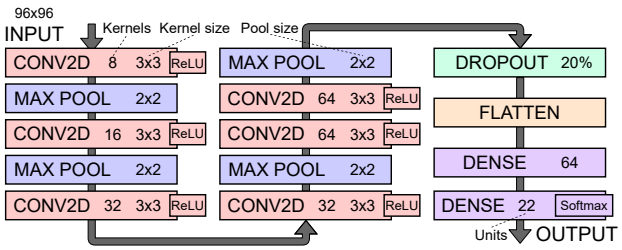


Figure 10. Schematic of a lightweight CNN model architecture. The low number of convolutional filters/kernels leads to very low number of trainable parameters in the network (only 200 000). However, this fact causes about 10 % worse model accuracy compared to a similar model with more filters, and more unstable results, as the validation accuracy curve very fluctuates each training epoch. Therefore, using such a low number of kernels may be inappropriate.

After that, I experimented with the number of fully connected layers and their units. Unfortunately, the outcomes were all quite similar to each other (the best ones did not stand out much), meaning that this testing brought out hardly thrilling results. I only found out that two or three dense layers might be a reasonable number.

Apart from all those relatively basic models, AlexNet (Fig. 2) and VGG-16, which far outweigh the others by complexity and number of training parameters, were tested too. The VGG-16 model structure turned out to be probably too complicated for my unvaried data, as the network was not able to learn anything. The AlexNet, having about 80 M parameters less than VGG-16, showed itself in a better light as the validation accuracy reaches up to 80 % (see Figure 11). Nevertheless, these results are still worse than the best ones achieved by the simpler models (as seen in Figure 14).

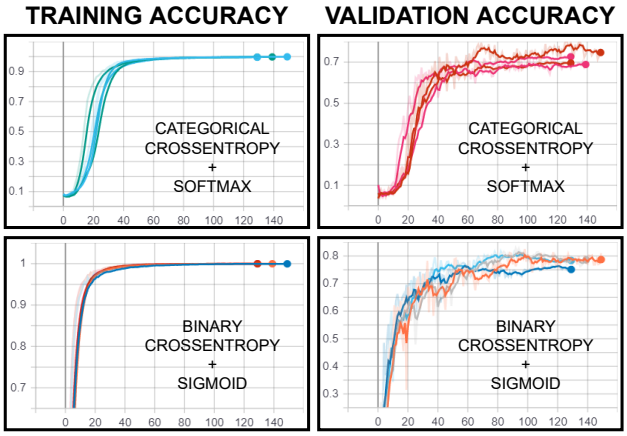


Figure 11. Visualization of the experiments made with the AlexNet CNN model. Each diagram groups results of a few measurements together. The horizontal axis represents training epochs, while the vertical axis represents training or validation accuracy. The AlexNet originally uses the softmax activation function at the output layer, but the sigmoid (combined with an appropriate cross-entropy loss function) was tested out of curiosity too. The model using the *sigmoid* variant reaches slightly better results. It is clear to see that the network is trained perfectly in just about 50 epochs, but the validation accuracy does not exceed 80 %. Apparently, it does not generalize well at all.

Data Augmentation Techniques Data augmentation proved itself to be very useful in fighting the model overfitting. Due to this, I widely experimented with all the individual transformation techniques (rotations, color changes and cropping), in order to set their parameters as precisely as possible. Firstly, I tested each of the techniques separately, and then I combined them together. For instance, I found out that the best results are achieved when rotating the image by 10° to 20° degrees in both directions, or when the brightness of a picture is changed just minimally. Another experimental results (presented in Fig. 12) showed how much should be the image cropped in the central region.

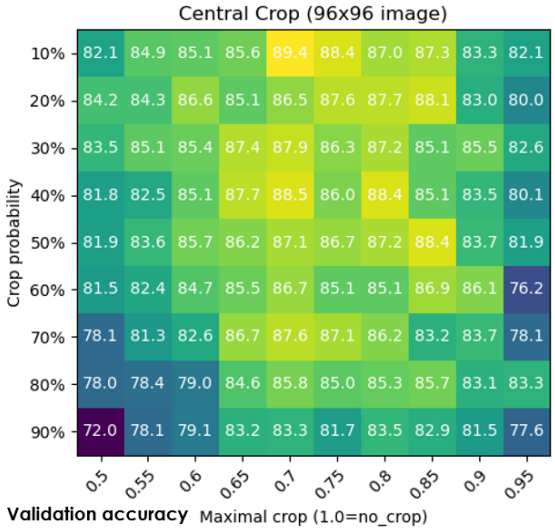


Figure 12. A heatmap visualizing the results of an experiment with the central crop data augmentation technique. An image is cropped in its central region with a given probability. The individual heatmap values represent the average validation accuracy in the last 40 epochs of the model. The values on the vertical axis represent the chance of cropping all images in a batch, while the horizontal axis represents the values of how much are the images being cropped (1.0 stands for no crop, while 0.5 means that the outer half of an image in cropped out). The heatmap shows that leaving about 70% of the original image (cropping out 30% of the outer region) brings out the best results. Besides, it is clear to see that the crop probability does not affect the results much.

Yoga Poses Similarity The latest experiments done so far analyze which of the Yoga poses sees the network as similar to each other. A custom confusion matrix, showing only the wrong decision for each pose, was designed for this purpose (see it in Figure 13). The results show that all the three **"Warrior III..."** Yoga poses are often misclassified by each other. A similar trend can be observed by the **"Thunderbolt..."** poses. Both of the **High Plank** and **Low Cobra** are quite often classified as the **Four Limbed Staff** pose. Besides, the two **"Warrior I..."** Yoga poses seem interesting as they are quite often classified as each other, but hardly ever classified as the other poses.

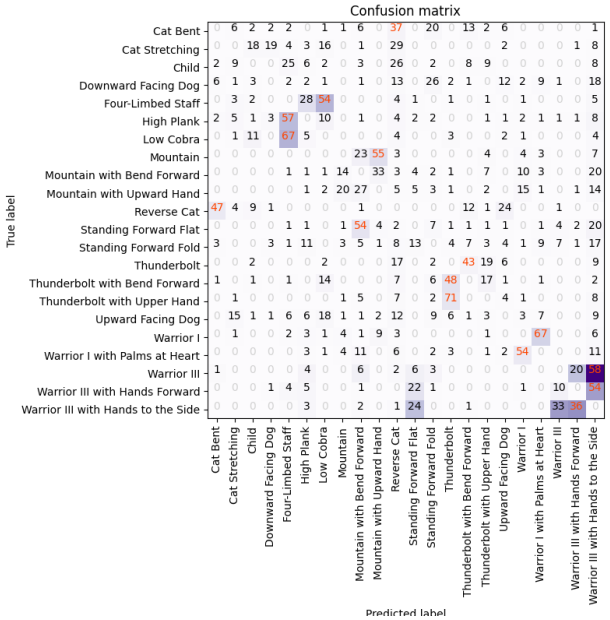


Figure 13. A confusion matrix, but is shows only the wrong decisions for each Yoga pose. Each value represents the number of how many times has a pose been misclassified by another one (the horizontal axis shows the predicted labels and the vertical axis shows the true labels). The aim of this visualization is only to show the possible similarity of the poses, not to measure how well are the poses being detected by a model. The results show that all the three **"Warrior III..."** poses are quite similar to each other and the same holds for the **"Thunderbold..."** ones. **High Plank** and **Low Cobra** are often classified as **Four Limbed Staff**, while the two **"Warrior I..."** poses are similar to each other, but unlike the others.

Learning Rate I experimented with the learning rate too. I originally used constant values for every epoch (approximately in the range of 2×10^{-4} to 7×10^{-3}), but then I started operating with several partly custom-built algorithms updating the learning rate dynamically at the end of each epoch. Nearly all of these brought better results compared to the measurements made with the static values. The learning rate is one of the most important model hyperparameters, but finding its optimal values is a difficult task.

5.2 Achieved Results

For most of the experiments, the frames were resized to the 96×96 px, which is expected as a good compromise providing a decent image resolution and a reasonable file size at the same time, but the best results so far were achieved using the dimensions of 224×224 px.

The CNN Model, consisting of eight convolutional and five max pooling layers in total, together with three dense layers (Fig. 9), reached the 100 % training accuracy in a few tens of epochs, and demonstrated its capability to detect Yoga poses by achieving the 91 % validation accuracy (as shown in Figure 14).

However, the 91 % were achieved using a **"hard"** dataset, into which videos were selected precisely in order to make the model predictions as hard as possible. When testing the Model on an **"easy"** validation dataset, into which videos were chosen to be similar to the videos chosen to form the training dataset, the validation accuracy reached 95 %.

But, it is fair to say that the Model is able to detect Yoga poses with the 91 % accuracy when trained on this particular Dataset, because the results reached with the "easy" datasets may probably not be valid.

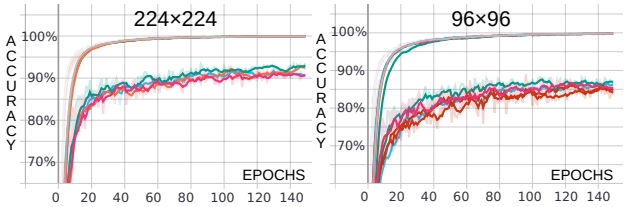


Figure 14. The training and validation accuracy of the CNN Model (9) visualized through the TensorBoard Scalar graphs. The graph on the right shows accuracy for a slightly modified version of the Model, where the input image dimensions are set to 96×96 px, instead of the original 224×224 px (graph on the left). Each of the graphs visualizes five independent measurements and for each of them, both the training and the validation accuracy are presented. Epoch numbers are showed on the horizontal axis and the accuracy is showed on the vertical one. The training accuracy, showed by the "smoother" curves, reaches the 100% value in both of the cases, meaning that the CNN Model learned perfectly on the training data. The validation accuracy presented reaches 91 % in the last 40 epochs for the original Model and 85 % for the lighter modification working with the 96×96 px input images. This Model was chosen as the best one among many others tested, because of the highest average validation accuracy during the last 40 epochs.

The *sigmoid* activation function coupled with the *binary cross-entropy* loss function, both of them involved in this measurement, confirm the fact of being such a powerful combination, at least for working with my Dataset.

6. Conclusions

The goal of this project was collecting Yoga videos and forming a dataset from them, as well as making decent predictions through a CNN on this data (Yoga pose detection).

In this paper, the process of forming the Dataset, including a custom video annotation tool or a frame capturing script, was described in detail, as well as the shape of the Dataset itself. Besides, the CNN model building and training process along with the techniques used was analyzed, step by step. In the end, the experiments together with the actual results achieved by an actual CNN model were demonstrated.

I experimented with more than a hundred CNN models, of which the most successful one (presented in Figure 9) detects Yoga poses with the 90 % accuracy (displayed in Figure 14). It implements the *sigmoid* activation function at the output layer instead of the *softmax* usually used for the multi-class classification.

However, the main contribution of this project are the tools for a dataset forming process and the Dataset itself. I collected 162 Yoga videos, designed an application tool providing an efficient way of creating video annotations and wrote a script capturing video frames along these annotations. The Dataset that I built contains 44 000 Yoga images of 22 different Yoga poses.

I believe that the annotation application is a practical and functional tool that anyone can use for video annotating if it fits their purposes. And in addition to that, some of the findings made by experimenting with CNN models might be also useful for somebody.

As for this project, the objective was attained. In the future I am planning to continue increasing the Dataset size as well as improving its diversity. Along with that, I aim to experiment with many more CNN models and various model parameters. Of course, anyone can use the results achieved and try to move them a bit further. The long-term goal remains designing a CNN model capable of detecting Yoga poses confidently even on a more complex dataset. The results already achieved show that it might be possible. Once done, the last step might be a smartphone app development and the CNN model implementation.

Acknowledgements

I would like to thank my supervisor Prof. Ing. Adam Herout, Ph.D., for guidance, motivation, the project idea itself and, finally, needful contributions to the Dataset forming process.

Computational resources were supplied by the project "e-Infrastruktura CZ" (e-INFRA LM2018140) provided within the program Projects of Large Research, Development and Innovations Infrastructures.

References

[1] Shih-En Wei, Varun Ramakrishna, Takeo Kanade, and Yaser Sheikh. Convolutional pose machines. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 4724–4732. IEEE Computer Society, 2016.

[2] Varun Ramakrishna, Daniel Munoz, Martial Hebert, James Andrew Bagnell, and Yaser Sheikh. Pose machines: Articulated pose estimation via inference machines. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, pages 33–47, Cham, 2014. Springer International Publishing.

[3] Santosh Yadav, Amitojdeep Singh, Abhishek Gupta, and Jagdish Raheja. Real-time yoga recognition using deep learning. *Neural Computing and Applications*, 31:https://link.springer.com/article/10.1007/s00521–019, 12 2019.

[4] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, November 1997.

[5] Z. Cao, G. Hidalgo, T. Simon, S. E. Wei, and Y. Sheikh. Openpose: Realtime multi-person 2d pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(1):172–186, 2021.

[6] M. U. Islam, H. Mahmud, F. B. Ashraf, I. Hossain, and M. K. Hasan. Yoga posture recognition by detecting human joint points in real time using microsoft kinect. In *2017 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, pages 668–673, 2017.

[7] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[8] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[9] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural

networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958, January 2014.

[10] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[12] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[13] Adrian Bradski. *Learning OpenCV, [Computer Vision with OpenCV Library ; software that sees]*. O'Reilly Media, 1. ed. edition, 2008. Gary Bradski and Adrian Kaehler.

[14] Martín Abadi et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[15] Bharath Ramsundar and Reza Bosagh Zadeh. *TensorFlow for Deep Learning: From Linear Regression to Reinforcement Learning*. O'Reilly Media, Inc., 1st edition, 2018.

[16] François Chollet et al. Keras. https://keras.io, 2015.

[17] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.