Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CENTER FOR PROJECT-BASED LEARNING

Department of Information Technology and Electrical Engineering

Spring Semester 2022

Visual Referee Detection on Nao Robots for RoboCup SPL 2022

Bachelor Thesis

Lukas Molnar
lmolnar@student.ethz.ch

June 2022

Supervisor: Dr. Seonyeong Heo, seonyeong.heo@pbl.ee.ethz.ch
Professor: Prof. Dr. Luca Benini, lbenini@iis.ee.ethz.ch

Acknowledgements

Firstly I would like to thank my supervisor Dr. Seonyeong Heo for always providing me with support and advice throughout this thesis. She offered up a lot of time to meet with me once or twice a week and helped with everything from debugging errors to preparing the whole data collection setup.

I would also like to thank Prof. Dr. Luca Benini and the Center for Project-Based Learning at ETH Zurich for giving me the opportunity to work on this thesis and gain an insight into the field of computer vision.

Further, a big thanks goes out to everyone who assisted in collecting the dataset for this thesis: Giuliano Albanese, Filippo Spinelli, Zichong Li, Kai Zhang, Tino Hämmerle, Damian Gerber, Pascal Ernst, Luca Tognoni and again Dr. Seonyeong Heo.

Finally, I would like to thank all members of the NomadZ RoboCup team for welcoming me and always helping out if I needed something to be explained for the codebase or the robots.

Abstract

At the 2022 edition of RoboCup a new challenge is being introduced, where Nao robots need to detect human referee signals in real-time. This thesis aimed to design a neural network model and implement it on a Nao robot to make accurate and quick predictions of the referee signals. To do this, the related work on human gesture recognition was researched, and the best performing models were implemented in Python using TensorFlow. These models consisted of convolutional as well as recurrent neural networks.

To train the models, a new referee signal dataset had to be collected based on the official description from RoboCup of the referee signals. The results from training showed that the best-performing models were the ones combining convolutional with recurrent layers, as opposed to a pure convolutional neural network. Also, the best-performing models used transfer learning by initializing the network with weights pre-trained on a larger dataset. These models achieved validation accuracies of 94% to 97%, meaning they were able to generalize well even though the collected dataset was relatively small, consisting of only 1409 total samples.

Finally, one of the best-performing models was implemented on a Nao robot by loading it as a TensorFlow Lite file and feeding the images received from the robot's camera into the neural network. The end result was successful since the robot was able to make live predictions every 1.5 seconds of the current referee signal, with an accuracy of over 90%.

Declaration of Originality

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor. For a detailed version of the declaration of originality, please refer to Appendix B

Lukas Molnar,
Zurich, June 2022

Contents

List of Figures

List of Tables

List of Acronyms

CNN	Convolutional Neural Network
GRU	Gated Recurrent Unit
LSTM	Long Short Term Memory
MLP	Multi-Layer Perceptron
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SPL	Standard Platform League
TfLite	TensorFlow Lite

Chapter 1

Introduction

RoboCup is a yearly international competition where teams compete by programming robots to perform various tasks. Initially the tournament started as a robot soccer competition, with the ambitious goal of having a team of robots play the FIFA World Cup champions by 2050 [6]. Since the launch of the RoboCup tournament, many more categories and leagues have been added to the original soccer competition. The main goal nowadays of the RoboCup tournament is to promote research in robotics and AI [6].

The NomadZ team of ETH Zurich competes in the Standard Platform League of RoboCup [7]. In this league each team programs the same humanoid robots, which are produced by Softbank Robotics and are called Nao robots [8]. No hardware modifications are allowed, so the performance of the robots is solely due to the written software of the team. The main part of the SPL competition is a 5 vs 5 soccer game between the Nao robots. However, each year several separate challenges are added which aim to expand the capabilities of the Nao robots. In the 2022 RoboCup tournament these challenges are: A 7 vs 7 Challenge, a Dynamic Ball Handling Challenge and a Visual Referee Challenge. The last one is the topic of this thesis.

1.1. Motivation

In the previous editions of RoboCup one thing that has not been tested is the detection of referee signals from a human referee. Under the current SPL rules [9], the robot is only required to listen directly to a human referee for the kick-off whistle. All other referee decisions are communicated to the robots via electronic game controller messages. In moving towards the 2050 RoboCup goal, robots will need to directly interpret referee calls and signals such as whistles, spoken calls and hand signals.

As a first step towards this goal, the 2022 RoboCup competition is introducing a Visual Referee Challenge. This challenge will take place separately from the regular 5 vs 5 game, with a pre-determined setup and 11 pre-determined referee signals. The robot needs to predict 5 of these signals and points are awarded for each correct prediction, as well as for how fast the prediction was made, if it was correct [9]. The full rules of the challenge as well as a description of the referee signals can be found in Appendix C.

Achieving a good result at RoboCup 2022 with the NomadZ team is the main motivation for thesis, however the broader motivation is to gain a better understanding of the computer vision techniques to detect human gestures and human actions. This topic has many real world applications such as sign language translation [10], which would help people who are hearing impaired and non-verbal to communicate with others who do not understand sign language. Another application for detecting general human actions is in autonomous driving, where it is important to distinguish between pedestrians, cyclist, skateboarders etc. in order to make good driving decisions.

1.2. Objective

The objective of this thesis is to implement a visual referee detection module on the Nao robot, in order to correctly predict the 11 referee signals that will be tested at RoboCup. The approach taken is to design a neural network model for gesture recognition and to train the model on a self-collected dataset of referee signals. Finally, the model must be deployed on the Nao robot to detect the referee signals in real-time.

To achieve the above mentioned objective, this project was split into three phases. Following are the main goals of each of these phases:

- **Phase 1:**
  Research what types of neural network models have been successful in detecting human gestures. Choose several of these models that are promising and implement them in Python using TensorFlow.
- **Phase 2:**
  Collect a video dataset of the referee signals that will be tested at RoboCup 2022. Train the models chosen in Phase 1 on this dataset and fine tune the model parameters to improve the performance of the models.
- **Phase 3:**
  Implement the models that performed best in training on the Nao robot. Use live image data from the robot's camera to feed into the neural network and make predictions of the referee signals in real-time. Quantify the accuracy and latency of the final implementation.

Chapter 2

Background

2.1. Convolutional Neural Networks

Convolutional Neural Networks (CNNs) are a special case of neural networks which use convolution instead of full matrix multiplication in the hidden layers [1]. An example of a regular neural network is the Multi-Layer Perceptron shown in Figure 2.1.

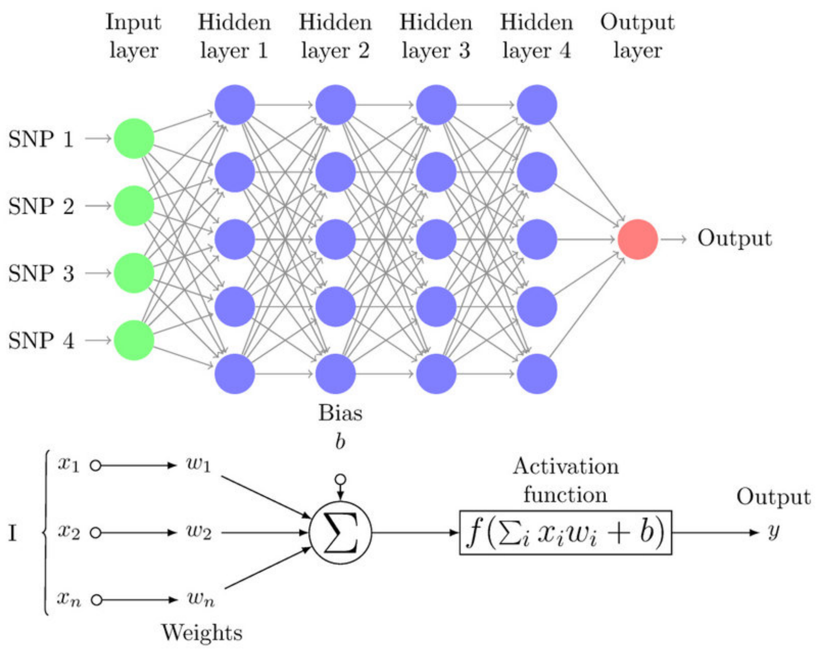


Figure 2.1.: Multi-Layer Perceptron (MLP) [1]

In the MLP the matrix (or vector) multiplications take place in each neuron, which are represented by a circle in Figure 2.1. The vector of outputs x_i from the previous layer is multiplied with the vector of weights w_i connecting the neuron to each neuron in the previous layer. A bias b is the added to this multiplication and fed into an activation function f, which produces an output y for each neuron (see Figure 2.1).

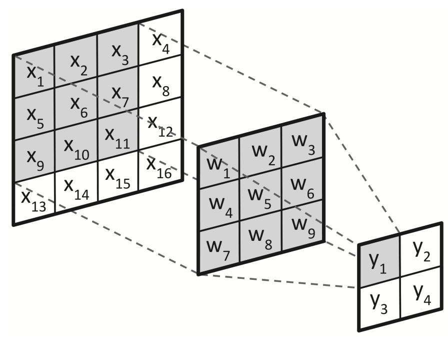


Figure 2.2.: Convolution Operation [2]

A convolution layer on the other hand applies a filter of weights only to a specific set of inputs, as can be seen in Figure 2.2. This requires the input data of a convolutional network to be structured, as for example images and videos are. In Figure 2.2 the input $\boldsymbol{X}$ is an image and the filter $\boldsymbol{W}$ slides over the image, producing an output value y_i for each position. This output value is the dot-product of the filter and the sub-matrix of $\boldsymbol{X}$:

$$y_1 = w_1x_1 + w_2x_2 + ... + w_9x_{11} \tag{2.1}$$

The weights in the filter $\boldsymbol{W}$ are what the network needs to learn. By choosing certain filters, the network can extract various spatial features such as horizontal or vertical edges. Since the filter slides over the whole input image it can extract the features no matter where they are located in the image. This is the main advantage of convolution layers for image recognition tasks over the MLP approach. To feed an image into a MLP such as in Figure 2.1, the input layer has to contain every pixel value of the image. In this case however, it is likely that the network will overfit to the training data since it will treat the same features differently if they are in a slightly different region of the image.

The following layers make up a standard CNN:

- **Convolution Layers:**
  These layers perform the convolutional operation described by Figure 2.2. The filters $\boldsymbol{W}$ which contain the weights are also called kernels, and are applied to the whole input of the previous layer in the network. This is called weight sharing, and significantly reduces the number of trainable weights in comparison to the MLP model that has dense interconnections between hidden layers. The kernel size has to be specified in each layer, for example it is (3x3) in Figure 2.2. The amount of kernels used also needs to be specified in each convolution layer, and is referred to as the number of channels of that layer. Lastly, the so called stride must be defined, which describes how large the shift is between two positions of the kernels. For example in Figure 2.2 the stride is (1x1), since the kernel moves over by one pixel in each of the input dimensions, to produce a new output value y_i.

As in the MLP model, an activation function f is usually applied to the output of each convolution layer. This activation function should add a non-linearity to the network, since the outputs y_i of the kernels are only linear combinations of the inputs x_i. One of the most popular activation functions, and the one mainly used in this paper, is the Rectified Linear Unit (ReLU) [1], which is defined as:

$$f(x) = max(0, x) \tag{2.2}$$

- **Pooling Layers:**
  Pooling layers help reduce the size of the network by substituting the output of the previous layer with a summary statistic of the neighboring outputs. Most commonly this is done by taking the maximum, mean or median of the neighboring outputs [1]. As in the convolution layers, the kernel size has to be specified. For example (2x2) means that each (2x2) block of the input is replaced by e.g. the maximum value of that (2x2) block. This means that the output tensor size is 25% of the input tensor size. Pooling layers do not contain any learnable parameters (weights), and they do not reduce the number of channels of the input layer.
- **Dropout Layers:**
  Dropout means that a given percentage of neuron outputs of a layer are set to zero. The percentage has to be specified and remains constant throughout the training process. However, the neurons that dropout is applied to are sampled randomly in each iteration. The reason for using dropout is to avoid overfitting [1].
- **Batch Normalization Layers:**
  A problem faced when training neural networks is that the distribution of each layer's inputs changes throughout the training process, as the parameters of the previous layers change. This slows down the training by requiring low learning rates and careful parameter initialization. In [11] Sergey Ioffe et al. refer to this phenomenon as "internal covariate shift", and address the problem by normalizing layer inputs for each training batch. According to their study this allows the use of much higher learning rates, and is less sensitive to parameter initialization. For this project the TensorFlow Keras layer `BatchNormalization` was used, which performs a transformation that maintains the mean output close to 0 and the output standard deviation close to [12].
- **Dense Layers:**
  The hidden layers in the MLP example (Figure 2.1) are called dense layers, since each neuron is connected to every neuron in the two neighboring layers. Dense layers are often also used in CNNs, and are placed after the convolution layers. These layers can process the feature maps produced by the convolution layers, forming an accurate output prediction [10].

2.2. Recurrent Neural Networks

In contrast to feed-forward neural networks such as CNNs or the MLP, a recurrent neural network (RNN) operates with time steps t. Each time step t is represented by a so called cell, a simple example thereof can be seen in Figure 2.3. Each RNN cell receives an input x_t and a state h_{t-1} coming from the previous cell in the time sequence. It produces an updated state h_t as well as an output o_t [2].

The following parameters are needed for the calculations of the RNN cell in Figure 2.3:

x_t: input vector

h_t: hidden layer vector

o_t: output vector

b_h, b_o: bias vectors

U_h, V_h, W_o: weight matrices

f_h, f_o: activation functions

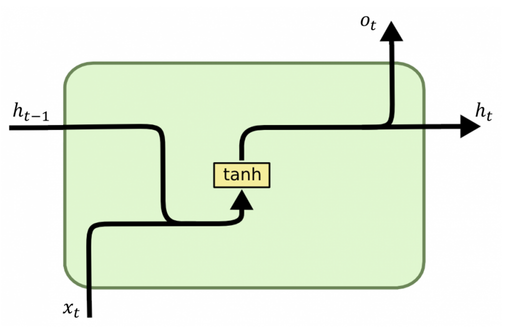


Figure 2.3.: Simple RNN cell [3]

With these parameters the state and output vectors are calculated in each time step as follows:

$$h_t = f_h(U_h x_t + V_h h_{t-1} + b_h) \tag{2.3}$$

$$o_t = f_o(W_o h_t + b_o) \tag{2.4}$$

The activation function used in Figure 2.3 is the hyperbolic tangent *tanh*. However, in general any activation function can be used.

The weight matrices and biases are what the network needs to learn during the training process. Due to the ability to store temporal information in the cell states, RNNs have shown success in a variety of problems such as speech recognition, language translation, and image captioning [13]. The types of RNNs discussed in this paper are Long Short Term Memory (LSTM) and the Gated Recurrent Unit (GRU), since they have shown success for various gesture recognition tasks (see chapter 3).

2.2.1. Long Short Term Memory (LSTM)

A limitation of simple RNN structures such as the one in Figure 2.3 is that for longer time sequences the cells tend to loose information. This is because in the calculations of the output and updated state, only the state from the directly preceding time step is taken into account. This leads to what is referred to as the short term memory problem [13].

The LSTM cell attempts to solve this problem by adding a state C, which stores long term information. Figure 2.4 shows the structure of a LSTM cell, with h being the short term memory state and x the input as before. The inside structure of the cell is split into three parts: the "input" gate which produces i_t, the "forget" gate which produces f_t, and the "output" gate which produces o_t (Figure 2.4). The gates themselves are made up of weight matrices as in the simple RNN example, and are trained to determine what the short term and long term features of the time sequence are. The exact mathematical expressions are not noted here, but can be derived analogous to the simple RNN example. The σ in Figure 2.4 represents the sigmoid function, which is used as an activation function like *tanh* or ReLU and is defined as:

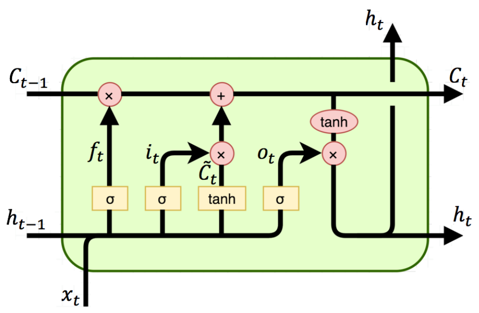


Figure 2.4.: LSTM cell [3]

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.5}$$

2.2.2. Gated Recurrent Unit (GRU)

GRU uses a similar approach to LSTM in order to solve the problem of long term memory loss in RNNs. The difference is that GRU does not add an extra state to the cell, as can be seen in Figure 2.5. Instead the GRU cell uses an "update" and a "reset" gate to extract the desired temporal features of the input sequence. The update gate produces z_t and determines the amount of previous information that needs to pass along to the next state. The reset gate produces r_t and determines how much of the past information should be neglected [13].

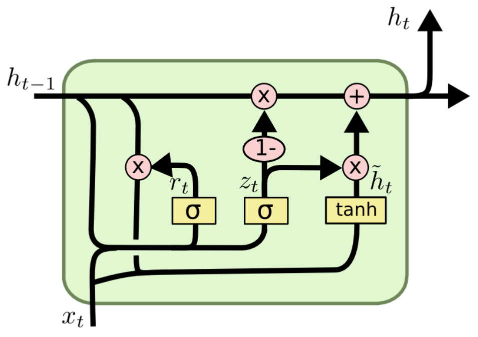


Figure 2.5.: GRU cell [3]

2.3. Training Network Weights

With the introduced architectures of convolutional and recurrent networks, the weights (also referred to as parameters) of these networks need to be learned, to produce a desired output for each given input. This is done by feeding a large amount of training data to the network, and updating the weights iteratively so that they minimize a certain loss function. The loss function used in this paper is Categorical Cross-Entropy which is a standard choice for classification tasks in general [14].

Categorical Cross-Entropy Loss is defined as:

$$Loss(y, \hat{y}) = -\sum_{i} y_i \cdot log(\hat{y}_i) \tag{2.6}$$

where y is the vector of desired outputs for the given input, and $\hat{y}$ is the output produced by the network. For classification tasks with a given number of classes, it is desirable to have the output vector consist of the probabilities the network predicts for each possible class. This is achieved by applying the Softmax activation function to the final layer in the network [10].

Softmax is defined as:

$$Softmax(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}} \tag{2.7}$$

where z_i are the individual outputs of the neurons in the last layer of the network, and K is the number of classes. It can easily be seen that the K softmax values of the output layer sum to a value of one. This lets each output z_i be interpreted as the probability the network predicts for class i.

The way the weights are updated to minimize the loss function is through backpropagation and gradient descent. Backpropagation is a method that propagates the error backward through the network, allowing the gradients of the weights in the previous layers to be calculated [1]. Using a gradient descent algorithm, the weights are then updated in each iteration. This is done by moving the weight values in the direction that decreases the loss most, based on the calculated gradients.

Chapter 3

Related Work

The goal of Phase 1 of this project is to research the state of the art neural network models in human gesture recognition, and see how these can be applied for the task of detecting referee signals. This chapter discusses the related work on human gesture recognition tasks in general, and what model architectures have been successful on various datasets. It is important to identify what kind of gestures the datasets are made up of and if they are similar to the referee signals used in this project. Furthermore, the models which perform well on small datasets should be identified, since the referee signal dataset needs to be collected from scratch and will hence be relatively small. From the most promising models the ones chosen for this project are described in more detail in chapter 5.

3.1. 3D Convolutional Neural Networks

For the task of classifying human gestures one needs to analyze video sequences. Since 2D convolutional neural networks (CNNs) have achieved great results for image recognition tasks, the main approach to video recognition tasks has been to use 3D CNNs [4]. The extra dimension in the convolution is needed since a video consists of a stack of frames, so the 3D convolution is applied to the 2 spatial dimensions of each frame, as well as the time dimension between frames. Performing a 3D convolution on a frame sequence extracts spatio-temporal features from the sequence. The spatial features are for example the edges in single frames, like one would see when applying a 2D CNN to one of these frames. The temporal features on the other hand find changes in time in a certain pixel region. This is important for identifying where and when movement is occuring in the sequence of frames. In the following subsections some implementations of 3D CNNs with varying depths are discussed and compared.

3.1.1. Deep 3D CNNs

One study analyzed the performance of various deep 3D CNNs on the following action recognition datasets: UCF-101, HMDB-51, ActivityNet and Kinetics [4]. These datasets consist of videos of humans participating in various activities from doing sports to playing musical instruments. The reason the results of this study were considered, is that many of the public "gesture" datasets focus on the human's hands and what signs they are making. However, for this project the full body of the referee needs to be in frame, as is mostly the case for the humans in the action recognition datasets.

The main goal of study [4] was to determine whether these action recognition datasets have enough data to train deep 3D CNNs. The specific architectures that they used were based on the ResNet architecture, which uses skip connections. An example of a typical ResNet block with a skip connection can be seen in Figure 3.1. "BN" stands for Batch Normalization and "ReLU" is the Rectified Linear Unit activation function (see chapter 2). The study analyzed ResNet architectures with depths ranging between 18 layers and 200 layers. For this project mainly the results of the 18 layer deep network ResNet-18 were analyzed, since much deeper architectures would not be feasible to run on the Nao robot. ResNet-18 consists of 17 convolutional layers, configured in blocks as in Figure 3.1, as well as one dense layer.

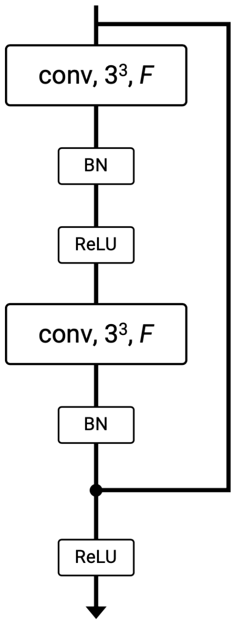


Figure 3.1.: ResNet Block [4]

The study concluded that ResNet-18 shows significant overfitting for the UCF-101, MBDP-51 and ActivityNet datasets. But for the Kinetics dataset, which is significantly larger than the other ones, such levels of overfitting did not occur. This shows that to train deep 3D CNNs one needs large amounts of data, which is a problem for this project since the collected dataset will be relatively small. However, what the study [4] also showed was that when pre-training the deep 3D CNNs on the Kinetics dataset, the models had significantly better results for the UCF-101 and HMDB-51 datasets, as seen in Table 3.1. So in order to use a deep 3D CNN for the dataset in this project, the network needs to be pre-trained on a larger dataset, ideally with gestures similar to the referee signals.

Model	**UCF-101 [15]**	**HMDB-51 [16]**
ResNet-18 (scratch)	42.2%	17.1%
ResNet-18 (pre-trained)	84.4%	56.4%

Table 3.1.: Validation Accuracy of ResNet-18 on Action Recognition Datasets [4]

3.1.2. Shallow 3D CNNs

There have also been approaches that use shallow 3D CNNs for gesture recognition. An example is an implementation of an Indian sign language recognition model [10]. This method showed promising results on a relatively small dataset with 2400 samples and 20 classes. The model consisted of only three 3D convolution layers, as well as three dense layers to process the features extracted by the convolution layers. Even though the whole bodies of the humans performing the gestures are not in frame at least their upper bodies are, and the humans are always facing the camera. This makes the dataset reasonably appropriate since the referee signals will also have the human facing the robot.

Based on the results of the Indian sign language recognition model, it was chosen as one of the models to be implemented and tested on the referee signal dataset. The exact architecture used for the referee signal dataset, and how it had to be slightly modified from the model used for the sign language dataset [10] is described in chapter 5.

3.2. Recurrent Neural Networks

Another common approach in gesture recognition tasks is to incorporate recurrent neural networks (RNNs) since they can identify the temporal behaviour of an input sequence such as a video. For gesture recognition the most commonly used types of RNNs in practice are LSTM and GRU, which have been explained in chapter 2. Two implementations are presented in the following sections that combined these RNN types with CNNs to detect human gestures.

3.2.1. Combining 2D CNNs with LSTM or GRU

One study used LSTM and GRU in combination with a 2D CNN to detect hand gestures for smart-TV applications [17]. The proposed model first applies the 2D CNN to each individual frame in the input sequence, to extract the spatial features. These feature maps are flattened into a 1D vector and fed into a recurrent layer (either LSTM or GRU) to extract temporal features as well. After the recurrent layer, a dense layer is added to combine all the features.

The dataset used in the above mentioned study is relatively small, consisting of 763 samples and 5 hand gesture classes (static as well as dynamic gestures). This suggests that the results of the study could be useful for training models on the referee signal dataset, since it will also be small and consist of rather few classes (11 signals in total). The results of the study state that the 2D CNN model with GRU had the highest validation accuracy, reaching 93%. The proposed model used transfer learning in the 2D CNN part, which means pre-trained weights were loaded into the network during initialization. This was done by using the TensorFlow Keras built-in MobileNet architecture for the 2D CNN part, for which one can load weights that were pre-trained on the ImageNet dataset [18]. The study also tested 3D CNN architectures with various depths on the dataset and found that the 2D CNN with GRU outperformed all of them. For this reason the 2D CNN with GRU model was chosen as the second model that would be implemented in this project and tested on the referee signal dataset. The exact architecture is described in chapter 5.

3.2.2. Combining 3D CNNs with Convolutional LSTM

The final approach that has been used for classification tasks of more complicated gestures is to use 3D CNNs in combination with convolutional LSTM [5] [19]. The reason for doing this is that the 3D CNNs extract local spatio-temporal features, which can be fed into the LSTM layer to produce global spatio-temporal features. This was missing in the example with 2D CNNs since there only the spatial feature maps of each frame were fed into the LSTM or GRU layers. Hence the local temporal features, such as movement in a certain pixel range, should be more difficult for the network to learn. Using the new approach the network can extract local spatio-temporal features through the 3D CNN layers as well as global features through the LSTM layers.

One study implemented the described approach using a Res3D architecture for the 3D CNN part of the network. This architecture uses skip connections like ResNet-18, which has already been discussed. Instead of a regular LSTM or GRU recurrent layer, the study used a convolutional LSTM layer, which can take 2D feature maps as inputs unlike a regular LSTM network. After the recurrent layer the study fused the local and global spatio-temporal features using a 2D CNN Mobilenet architecture. The summary of the network layout is show in Figure 5.3.

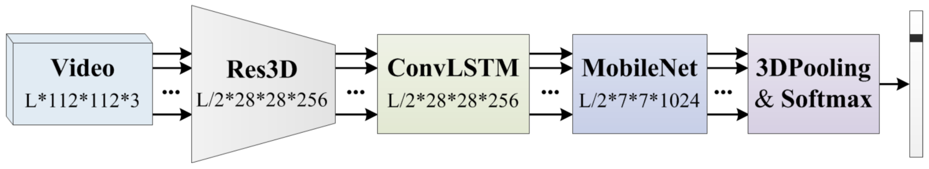


Figure 3.2.: Res3D with Convolutional LSTM and MobileNet [5]

The dataset used is called IsoGD, with a large amount of 41,662 videos in total for training and validation [20]. The dataset consists of 249 static and dynamic gestures, with the upper body of the human always being in frame. Some of the gestures are very similar to the referee signals, which is why this model was chosen to be tested on the referee signal dataset. However, as explained in subsection 3.1.1 deep network topologies perform significantly better on small datasets when pre-training the models on larger datasets. So for this project the IsoGD dataset was chosen to pre-train the model, after which it could be trained on the smaller referee signal dataset (see chapter 5).

Chapter 4

Dataset

Since this is the first time that referee signals are being tested at RoboCup, there is no existing dataset of the required signals. For this reason a new dataset had to be created from scratch, based on the description of the signals in the RoboCup rule book [9]. This chapter explains how the dataset was collected using the Nao robot and how the images were processed to be used for training the models.

4.1. Data collection

4.1.1. Setup

The same setup which is described in the RoboCup rule book [9] was used to collect the referee signal dataset. The full description can be found in Appendix C. In summary, the position of the robot is in the center of the soccer field, facing the sideline. The referee is standing on the sideline, facing the robot and performing the signals. In order to train the models on various backgrounds instead of only the one wall in the lab, a green screen was mounted behind the referee. Using different backgrounds in the dataset should train the models to work on any background, which is important since there is no description for how the background scene will look like at RoboCup.

In total 10 volunteers assisted in collecting the referee signal dataset. They wore red gloves, since the RoboCup rules for this challenge state that the referee at the competition must wear read gloves [9]. In regards to clothing the rules state that the RoboCup referee will wear a black and white striped jersey "if available". To be sure that the model can also predict the signals in the case where the referee is not wearing such a jersey, the subject were given a free choice of clothing.

4.1.2. Procedure

The subjects performed each referee signal for several seconds, while the image data was transferred from the robot to a local machine. This was done by using the module `ImageAcquisition` in the robot's codebase, which stored the images received from the robot's camera at a certain frame rate. The images could then be copied to the local machine using SCP (Secure Copy Protocol). For the frame rate 10 Hz was chosen, since this is the frame rate used in the IsoGD dataset mentioned in chapter 3. One of the models uses transfer learning with weights pre-trained on this dataset, so it is beneficial to use the same frame rate in the new referee signal dataset.

The description of the gestures in the RoboCup rule book [9] is not very specific (see Appendix C), in that there is no statement on the exact rotation of the hand when the arm of the referee is extended. This is why each subject performed all of the gestures twice, once with the palm of the extended hand facing the robot, and once with the extended hand rotated 90 degrees so the robot sees the hand from the side. This gives the dataset more variety and should let the model make better predictions no matter how the hand of the referee is rotated.

4.2. Image Processing

In order to train the neural network models on different backgrounds, the green screen in the image received from the robot had to be replaced by various background images. In total 34 background images were used, with various lighting conditions and background settings. Images were chosen with and without people in the background to create further variety in the training set.

The following steps had to be taken to replace exactly the right background pixels, as well as to denoise and resize each image:

1. **Find Green Pixel Range:**
   To replace the green pixels in each image with a new background, first the range of green pixel values had to be determined. The exact range fluctuated on the different days when data was collected, because of small differences in the lighting condition of the room. An approximate general range is given in Table 4.1.
2. **Additional Green Requirement:**
   Since there were always some colors within the given pixel range that did not belong to the green background, an extra condition was added that the green value of each pixel must have a value 20-40 units higher than the red and blue values of the pixel. This condition also fluctuated a bit for the different days when data was collected, and hence had to be fine tuned each time.

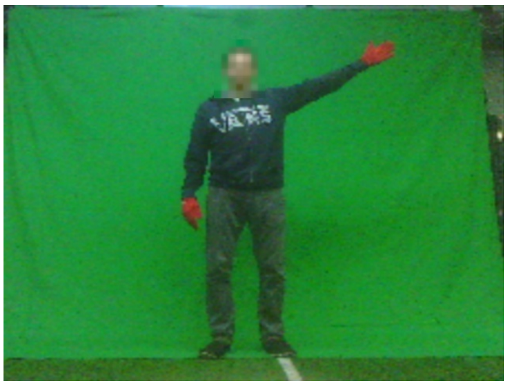
	Lower Limit	**Upper Limit**
Red	0	130
Green	80	230
Blue	0	140
Color	🟩	🟩

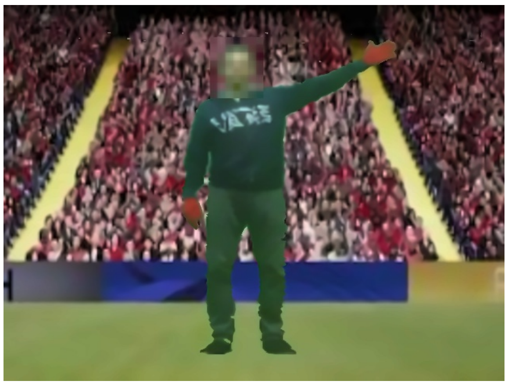
Table 4.1.: Pixel Range for Green Screen Replacement

3. **Replace Green Screen:**
   Now all pixels that are within the green range from step 1 and fulfil the extra requirement in step 2 can be replaced by the pixels of the new background image. When loading the background image, it was cropped randomly to prevent the network from overfitting to the exact positions of the referee in respect to the background scene. An example of an image before and after replacing the green screen can be seen in Figure 4.1.

4. **Denoise Image:**
   Since the image quality of the robot is not very high there are usually small patches of pixels that belong to the background but were not replaced in the last step. These pixels usually had similar values to pixels that were part of the subject performing the referee signal, so the green range could not just be adjusted to include them without erasing parts of the referee subject. To solve this problem the OpenCV function `fastNlMeansDenoising` was used, which performs a non-local means denoising algorithm on the image. Like this most of the remaining green patches were smoothed out to become part of the background.

5. **Resize Image:**
   Since the input tensor of the neural network takes images with pixel dimensions 112x112, the last step was to resize the images to these dimensions. This also helped with smoothing out the remainder of the green patches which the denoising algorithm was not able to process. The final result for an example image can be seen in Figure 4.1.


(a) Image from Robot


(b) Green Screen replaced

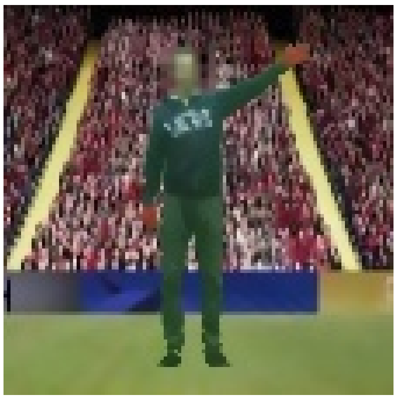

(c) Denoised and Resized

Figure 4.1.: Image Processing Steps

4.3. Finalizing the Dataset

To feed the image data into the neural network a fixed sequence length had to be used. The choice was made to use 15 frames for the sequence length in order for the robot to make a prediction of the referee signal every 1.5 seconds (frame rate 10 Hz). The reason why 15 frames where chosen instead of a smaller number for faster predictions is that when analyzing the referee signals there is the risk of making a false prediction to quickly for certain signals. For example Figure 4.2 shows two signals "kick-in left" and "corner-kick left". One can suspect that while the referee is getting in position for "kick-in left" the extended arm will pass through the position of "corner-kick left" for a few frames. This causes a risk of predicting the wrong signal too soon if the number of frames per sequence is chosen too low.

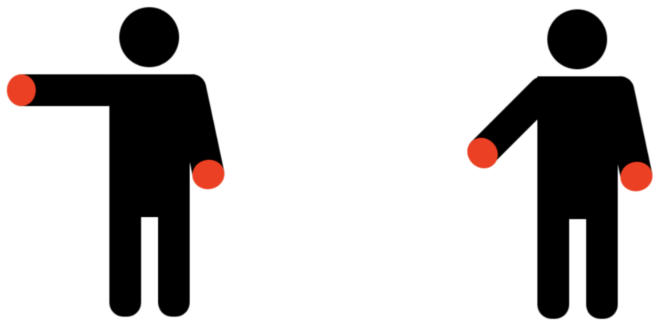

(a) Kick-in left (b) Corner-kick left

Figure 4.2.: Similar Referee Signals

The finalized dataset contains 1,409 samples of 15 frame sequences. These samples were split into training and validation sets, with different referee subjects and different backgrounds used in each set, in order to detect if the models are overfitting to the training data. In Table 4.2 a summary of the dataset split is shown. The dataset is not made public, due to the fact that other teams competing at RoboCup could make use of it.

	Training Set	**Validation Set**	**Total**
Samples	1,190	219	1,409
Referee Subjects	7	3	10
Backgrounds	28	6	34

Table 4.2.: Referee Signal Dataset Split

Chapter 5

Implementation

From the findings of the related work in chapter 3 three model concepts were chosen to be implemented for the referee signal detection task. This chapter presents the three model architectures and how they were trained on the new referee signal dataset. The implementation of the trained models on the Nao robot to detect the referee signals in real-time is also described.

5.1. Implemented Models

The three models chosen from chapter 3 are the following: 3D CNN (shallow implementation), 2D CNN with GRU and 3D CNN with convolutional LSTM. The following sections describe the structure of each of these models in detail, and how the implementations discussed in chapter 3 were adapted to run on the referee signal dataset. Table 5.1 displays a summary of the number of trainable and non-trainable parameters (weights) for each model, in their new implementations for the referee signal dataset.

Model	**Trainable Parameters**	**Non-Trainable Parameters**	**Total Parameters**
3D CNN	2,412,428	1,728	2,414,156
2D CNN with GRU	3,428,172	23,936	3,452,108
3D CNN with ConvLSTM	1,935,244	8,269,376	10,204,620

Table 5.1.: Model Parameter Summary

5.1.1. 3D CNN

The 3D CNN model chosen to be implemented for the referee signal dataset is based on the shallow 3D CNN model discussed in chapter 3. The motivation being that this model showed success in predicting Indian sign language gestures on a relatively small dataset of 2400 samples [10]. A summary of the model as implemented for the referee signal dataset can be seen in Figure 5.1, with the tensor dimensions noted for each layer. A list of each individual layer can be found in Appendix D, including how many weights each layer contains. Following is an explanation of Figure 5.1 with the corresponding parameter choices for each layer.

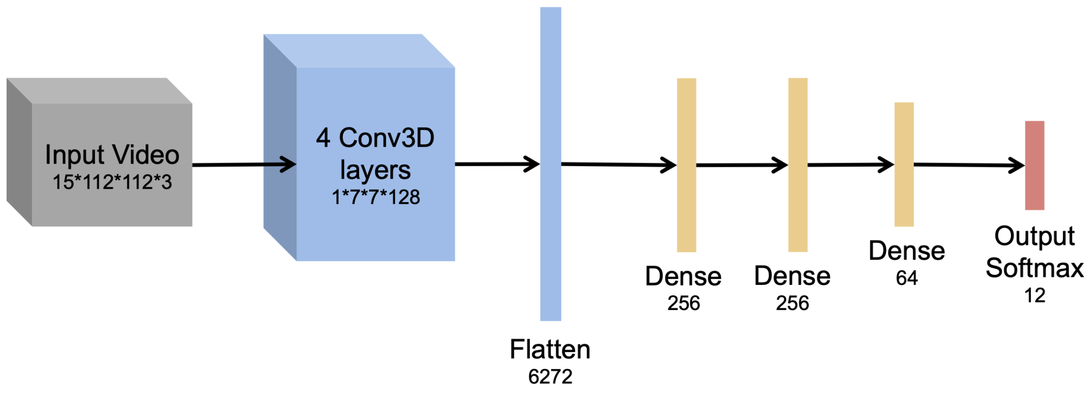


Figure 5.1.: 3D CNN Model

For clarity, the input tensor dimensions in Figure 5.1 are noted as 15*112*112*3 since the input to the network is a sequence of 15 frames, each with dimensions 112*112, and 3 color channels (RGB). The model then consists of four convolution layers, using kernel size (3x3x3) and stride (1x1x1). Padding is added to the convolution layers, so the input dimensions are not reduced. In-between each convolution are the following layers: ReLU activation, batch normalization and max pooling with a (2x2x2) pool size. The last max pooling layer has to use a (1x2x2) pool size since the first dimension has been reduced to size one at this point. This can be seen in Figure 5.1, where the output tensor of the four convolution layers has size 1*7*7*128.

After the four convolution layers the tensor of the created feature maps is flattened and fed into a sequence of three dense layers containing 256, 256 and 64 neurons. The dense layers use ReLU activation and after the first two dense layers dropout of 30% is applied. After the last dense layer the output layer with Softmax activation is added. There are 12 classes in the output layer, since on top of the 11 referee signals a class "no signal" was added, where the referee is standing still. This was done since according to the description of the RoboCup challenge this will be the initial stance of the referee at the competition (see Appendix C).

Finally, in order to reduce overfitting L1 regularization was applied to the convolution and dense layers in the 3D CNN model, as proposed by the implementation of the Indian sign language detection module [10].

5.1.2. 2D CNN with GRU

The second model is based on an implementation of a 2D CNN with GRU for hand gesture recognition, as described in chapter 3. This implementation showed better results than various 3D CNN models on the same dataset, which is why it was chosen to be implemented in this project. Since a 3D CNN was also implemented in this project, the results of these two models can be compared to see if for the referee dataset the 2D CNN with GRU also shows better results. Figure 5.2 displays a summary of this model in it's implementation for the referee signal dataset. In Appendix D again a list of all layers can be found, with the number of weights in each layer. The following section explains Figure 5.2 in more detail, and how the model extracts the spatial and temporal features of the input frame sequence separately.

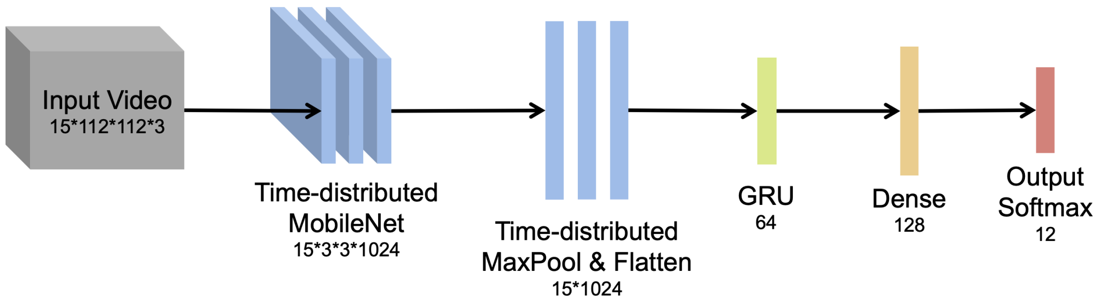


Figure 5.2.: 2D CNN with GRU Model

The 2D CNN part of the network is made up of the TensorFlow Keras built-in MobileNet architecture, and is initialized with weights pre-trained on the ImageNet dataset [18]. This 2D CNN produces spatial feature maps of size (3x3) with 1024 channels for each frame in the input sequence. Since the 2D CNN should be applied to each individual frame of the input sequence, it is added to the model using the Keras layer `TimeDistributed`. Max pooling with kernel size (3x3) is then applied to the 2D CNN feature maps and they are flattened to produce 1024 feature vectors for each frame. These vectors are then fed into the GRU network, consisting of 15 cells since there are 15 frames. The GRU network produces one single output vector of size 64 which extracts the temporal features of the frame sequence. A dense layers with 128 neurons and ReLU activation is added to combine all features. Finally, the output layer with Softmax activation terminates the network.

5.1.3. 3D CNN with Convolutional LSTM

The final model is based on the implementation of a 3D CNN with convolutional LSTM for gesture recognition on the IsoGD dataset [5], as discussed in chapter 3. Figure 5.3 displays a summary of this model in it's implementation for the referee signal dataset. Since the network is very large, the list of all layers is not added to the appendix. However, the code from the original study is publicly available on github [21], where the full model architecture can be found.

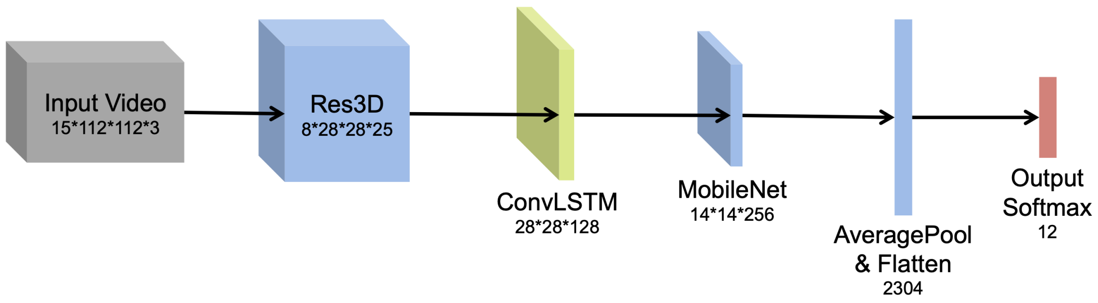


Figure 5.3.: 3D CNN with Convolutional LSTM Model

Following is a description of the main parts of the network displayed in Figure 5.3. Additionally it is explained how some layers in the network were adapted from the original implementation to train the model on the referee signal dataset. The model consists of three main parts:

1. 3D CNN (so called Res3D) for local spatio-temporal feature extraction
2. Convolutional LSTM for global temporal feature extraction
3. 2D CNN (MobileNet) for feature fusion

The Res3D part was taken one to one from the implementation in the study [5]. It is a 3D CNN architecture which utilizes skip connections, similarly to the ResNet architectures discussed in chapter 3. For the convolutional LSTM layer the study created some custom functions which adapted the convolution operations within the LSTM cells. However, for this study the regular Keras layer `ConvLSTM2D` [22] was used. Since the referee dataset is much smaller than the IsoGD dataset, only one `ConvLSTM2D` layer was used instead of two, to reduce the chance of overfitting. Also, for the same reason a smaller version of the 2D CNN MobileNet was used in the third part of the network. The purpose of this 2D CNN is to fuse to local and global features extracted by the Res3D and ConvLSTM parts of the network. Finally, average pooling with kernel size (4x4) is applied to the MobileNet output, upon which the tensor if flattened and passed to the network's output layer with Softmax activation (see Figure 5.3).

In chapter 3 it was stated that deep 3D CNN models perform significantly better on small datasets, when initializing the weights with ones pre-trained on a larger dataset. Since weights pre-trained on the large IsoGD dataset have been made public for the Res3D layers, these weights were loaded into the network to train on the referee dataset. In order to also load pre-trained weights for the other layers, the adapted model with one `ConvLSTM2D` layer and a smaller MobileNet was trained on the large IsoGD dataset, and saved as a Keras model. These weights could be loaded into the rest of the network when preparing to train on the referee signal dataset.

Since even with the smaller version of the model the number of parameters was still over 10 million, the weights in the Res3D layers were frozen during the training process. This means that they were initialized with the weights pre-trained on the IsoGD dataset, and not updated further while training the network on the referee signal dataset. Like this, the number of trainable parameters ended up being 1,935,244 and the number of of non-trainable parameters 8,269,376 (see Table 5.1).

5.2. Training Models on the new Dataset

The finalized dataset was made up of 1409 videos, each video being a sequence of 15 frames. The dataset was split into training and validation sets with the two sets containing different referee subjects performing the signals, as well as different backgrounds (see section 4.3). The neural network models were programmed in Python using TensorFlow. They were run on a remote server using a Nvidia Titan RTX GPU for improved performance.

Each model was trained for 25 epochs with a batch size of 16. For the loss function Categorical Cross-Entropy was used, as described in chapter 2. For the optimizer "Adam" was used, which is a gradient descent method that has proven to be very efficient when working with large problems involving a lot of data or parameters [23]. Adam adds a momentum term to the weight update function, which accelerates the gradient descent algorithm. It also makes use of Root Mean Square Propagation (RMSP), which is an adaptive learning algorithm that updates the weights based on the exponential moving average of the respective gradients [23].

To reduce the models from overfitting to the training data, three data augmentation techniques were implemented: random crop, random brightness augmentation and random contrast augmentation. By augmenting the training data randomly during each iteration of the dataset the model can learn to generalize better. This is especially useful in the case of a small dataset such as the referee signal one, since overfitting is a severe concern.

5.3. Implementing Models on the Nao Robot

The training of the implemented network models on the new referee signal dataset concluded phase 2 of this project. The results of which are described in chapter 6. The final phase consisted of implementing the best performing models on the Nao robot, in order for the robot to make predictions of the referee signals in real-time. The following sections describe how the trained models were saved and loaded onto the robot, as well as how the live camera data from the robot was fed into the network to make continuous predictions.

5.3.1. Loading Models as TfLite Files

After the training process was completed, the state of the network with the lowest validation loss achieved was saved as a TensorFlow Keras model. In order to implement this model on the Nao robot, it had to be converted to a TfLite (TensorFlow Lite) file. TensorFlow Lite is an open-source deep learning framework, which provides a set of tools that enables on-device machine learning. It allows neural network models to be run on mobile, embedded and IoT (Internet of Things) devices [24]. Once the model was converted to a TfLite file, it could be loaded onto the Nao Robot following the instructions in the TensorFlow Lite C++ API Reference [25].

First however, a module `TfLiteRefereePerceptor` and representation `RefereePercept` were defined in the robots code base. The representation only contains a streamable parameter `nr_frames` which stores how many frames have been loaded into the input tensor of the network. The module on the other hand is where the TfLite file for the neural network model is loaded and run. The main code for the module can be found in Appendix E.

The constructor of the module `TfLiteRefereePerceptor` loads the TfLite file and initializes the TfLite interpreter. Also, the constructor checks whether the input dimensions to the network are correct, and allocates an address in the robot's memory to the input tensor of the network. The remaining functions are used to run the neural network model in real-time.

5.3.2. Running Models in Real-Time

With the model loaded through the TfLite file the input tensor of the network has to be filled with live image data. Like this the robot can make continuous predictions of what referee signal it is seeing.

To load the input tensor a function `update` is called each time the robot receives a new image from it's camera (this occurs every 0.1 seconds). Then the received image is loaded into the input tensor using the the function `load_frame` (see code snippets in

Appendix E). Important to note here is that the image first needs to be resized to the correct network dimensions of 112x112, and converted to the right color space of RGB instead of YCbCr. The latter is used by default on the robot whenever an image is converted to a `cv::Mat` data structure, which needs to be done in this case to resize the image to 112x112. Furthermore, the sequence of pixel color values has to be set to BGR (blue, green, reed), since this is how the OpenCV function `imread` stores images in an array [26]. This function was used when reading the images from the collected referee signal dataset during training, so the pixel color sequence needs to be the same when loading the input tensor with live image data.

Once 15 frames are loaded into the input tensor, the `update` function calls a `predict` function (see code snippet in Appendix E). This function reads the output tensor of the network, which contains the probabilities for each referee signal. The *argmax* signal is then printed to the console along with it's probability predicted by the network. If the probability is over 90% the function `say_prediciton` (see Appendix E) is called and the robot announces the signal and which team it is being shown for (blue or red team).

After this, the parameter `nr_frames` is set to 0 by the function `refereePercept.reset()`, meaning that the process starts all over again and the input tensor to the network is overwritten until another 15 frames are received. Like this continuous predictions can be made every 1.5 seconds by the Nao robot, and they are announced if the confidence is over 90%. This confidence threshold was chosen based on the limited tests done in the lab, however more tests should be done before the RoboCup challenge in order to optimize when the robot announces it's final prediction. This is important since at the RoboCup challenge the robot has only one chance to make a prediction of the referee signal.

Chapter 6

Results and Discussion

To compare and evaluate the performance of the trained models, the collected referee signal dataset was split into training and validation sets. The validation set was chosen to have different people performing the referee signals, as well as different backgrounds, in order to detect more easily if the models are overfitting to the training set. In this chapter the results of the three trained models from chapter 5 are presented and discussed.

6.1. 3D CNN

The 3D CNN model achieved a maximum validation accuracy of 73.2%, with the training history displayed in Figure 6.1. The model is clearly overfitting, since the training accuracy is significantly higher thant the validation accuracy. This means the model is not generalizing well and makes many wrong predictions for new referee subjects with new backgrounds.

These results indicate that the proposed shallow 3D CNN model is not suitable for the collected referee signal dataset. Referring back to chapter 3, this model was based on the successful implementation of a shallow 3D CNN for Indian sign language recognition [10]. That implementation used a dataset of size 2400, which is larger than the referee signal dataset, but small in comparison with other gesture datasets such as IsoGD [20]. Hence, one can conclude that this shallow 3D CNN implementation is not suitable for all small human gesture datasets.

6.2. 2D CNN with GRU

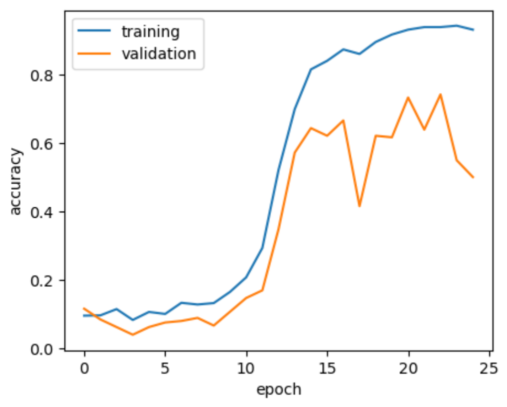
The 2D CNN with GRU model achieved a maximum validation accuracy of 94.6%, with the training history displayed in Figure 6.1. Here there is practically no overfitting, with the validation accuracy being very close to the training accuracy throughout the training process (see Figure 6.1). The confusion matrix for this model is displayed in Figure 6.2 with the list of label abbreviations shown in Table 6.1. In the confusion matrix one can see that indeed there are only few false predictions made. The entries shown in the confusion matrix are the samples from the validation set. They sum to 219 since there are 219 samples in the validation set (see chapter 4).

One can conclude that this model performs very well on the referee signal dataset and notably better than the 3D CNN model. This is the same result as found in the study which proposed the 2D CNN with GRU model [17]. That study had the networks detect hand gestures for smart-TV applications and achieved a validation accuracy of 93% using the 2D CNN with GRU model. This is very close to the 94.6% achieved for the referee signal dataset, letting this model be classified as a success.

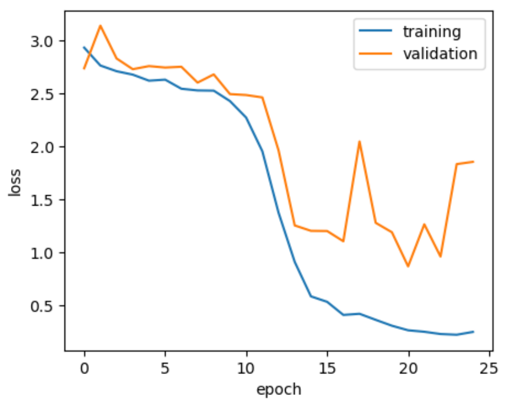
6.3. 3D CNN with Convolutional LSTM

The 3D CNN with Convolutional LSTM model achieved a maximum validation accuracy of 96.9%, with the training history displayed in Figure 6.1. Here there is also practically no overfitting, since the validation accuracy is nearing 100%. What is interesting though, is that the validation accuracy has some sharp changes in the initial part of the training process (see Figure 6.1). At this point the training accuracy is already around 95%, however the model is still able to improve the validation accuracy, achieving over 90% consistently for the last 10 epochs. A possible explanation for these fluctuations in validation accuracy is that the model has a very high number of parameters. Even though most of the parameters are non-trainable, the high total number could be the reason for the model taking more epochs until it is able to generalize well and achieve a stable high validation accuracy.

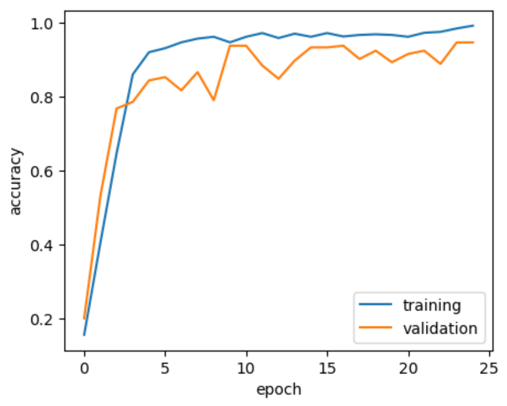
The confusion matrix for this model is displayed in Figure 6.2. As for the 2D CNN with GRU model there are very few false prediction, letting this model too be classified as a success.



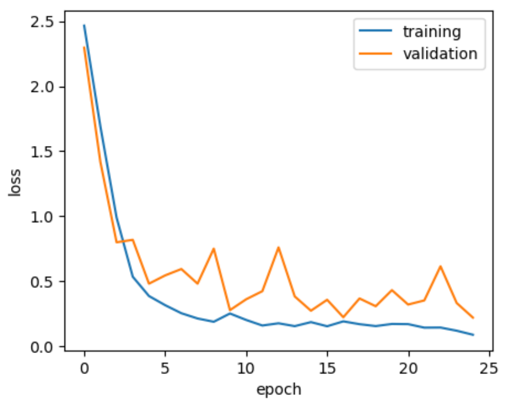
(a) 3D CNN - Accuracy



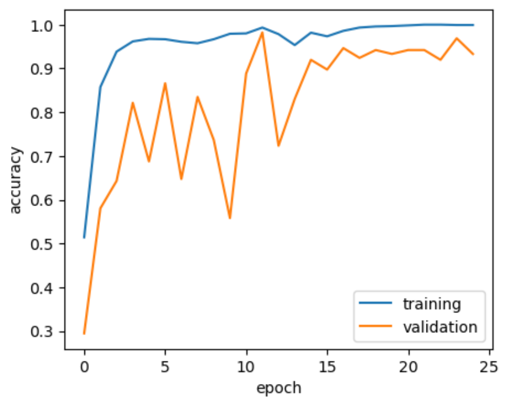
(b) 3D CNN - Loss



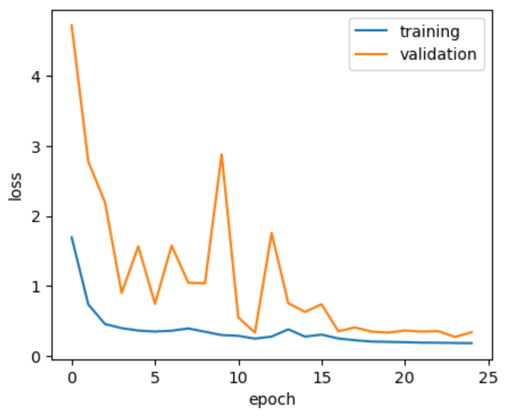
(c) 2D CNN with GRU - Accuracy



(d) 2D CNN with GRU - Loss

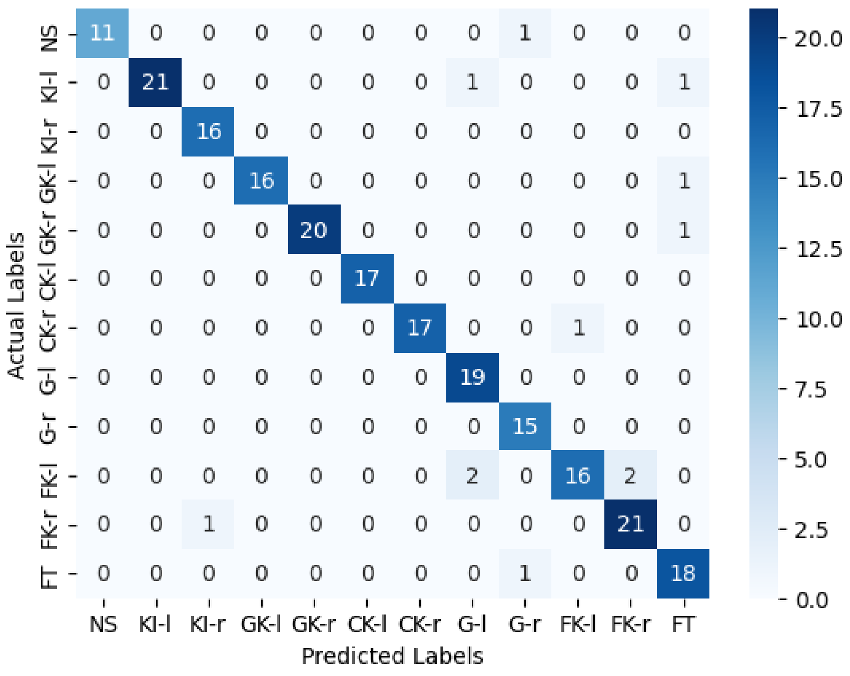


(e) 3D CNN with ConvLSTM - Accuracy

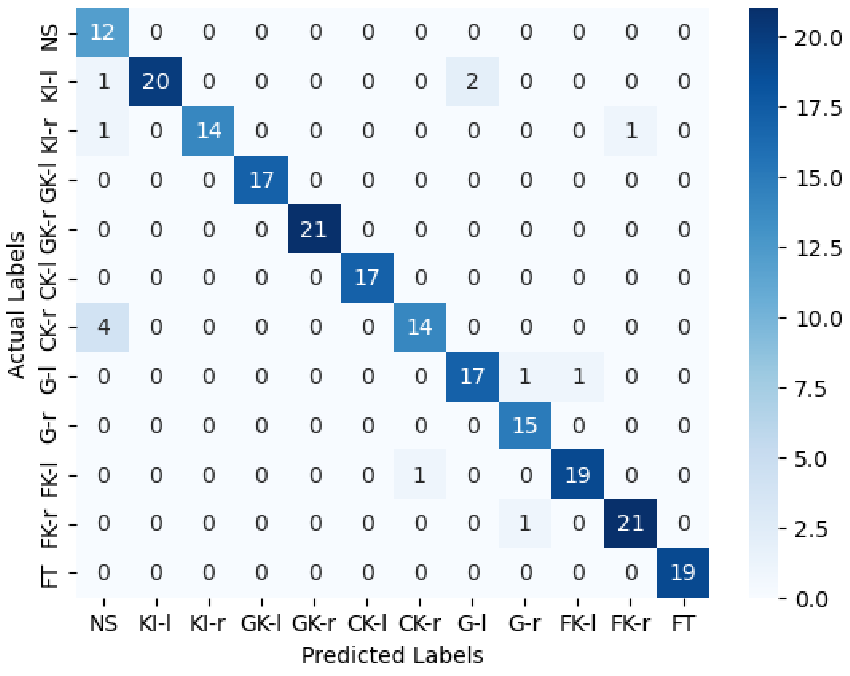


(f) 3D CNN with ConvLSTM - Loss

Figure 6.1.: Accuracy and Loss History for all Models



(a) 2D CNN with GRU



(b) 3D CNN with ConvLSTM

Figure 6.2.: Confusion Matrices for Best-Performing Models

NS	No signal
KI-l	Kick-in left
KI-r	Kick-in right
GK-l	Goal-kick left
GK-r	Goal-kick right
CK-l	Corner-kick left
CK-r	Corner-kick right
G-l	Goal left
G-r	Goal right
FK-l	Free-kick left
FK-r	Free-kick right
FT	Full-time

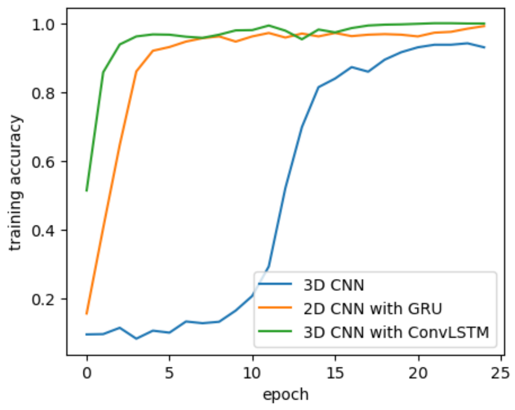
Table 6.1.: Abbreviations of Referee Signal Labels

6.4. Summary

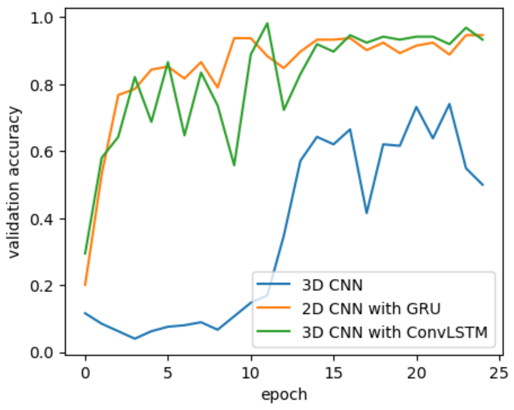
In summary, the best performing models were clearly the ones making use of recurrent layers, namely the 2D CNN with GRU as well as the 3D CNN with ConvLSTM. Table 6.2 lists the maximum validation accuracy achieved by each model and Figure 6.3 displays the histories of training and validation accuracy for all three models. One can clearly see how much quicker the accuracies increase for the models with recurrent layers over the 3D CNN model. The main reason for this is likely that these models utilize transfer learning by initializing the network with pre-trained weights. The 3D CNN on the other hand which uses random weight initialization takes significantly more epochs until training and validation accuracies start rising (see Figure 6.3). Additionally, once the accuracies of the 3D CNN have stabilized, the validation accuracy is significantly bellow the other two models. This lets one make the assumption that networks combining recurrent layers with convolution layers are more appropriate for the task of detecting referee signals than a pure convolutional neural network.

Model	**Validation Accuracy**
3D CNN	73.2%
2D CNN with GRU	94.6%
3D CNN with ConvLSTM	96.9%

Table 6.2.: Maximum Validation Accuracy for each Model



(a) Training Accuracy



(b) Validation Accuracy

Figure 6.3.: Accuracy Comparison of all Models

Chapter 7

Conclusion and Future Work

The goal at the beginning of this project was to implement a referee detection module on a Nao robot, in order to predict the 2022 RoboCup challenge referee signals in real-time. This goal was achieved by collecting a referee signal dataset from scratch and training neural network models on this dataset.

The results of the trained models showed that the best performing networks were the 2D CNN with GRU and the 3D CNN with ConvLSTM. These models both use a combination of convolution layers and recurrent layers to extract spatial and temporal features from the input frame sequence. The validation accuracies achieved by these models were 94.6% for the 2D CNN with GRU and 96.9% for the 3D CNN with ConvLSTM. The validation set contained different subjects performing the referee signals as well as different backgrounds to the training set. This shows that these models were able to generalize well and make accurate predictions on new data.

The third model which was tested showed a lower maximum validation accuracy of 73.2%, and took significantly more epochs for the accuracy to increase to this level. The suspected reason for the slow increase is that this model did not load pre-trained weights when initializing the network like the other two models did. This leads to the conclusion that using pre-trained weights, whether they were pre-trained on ImageNet for the 2D CNN with GRU or on IsoGD for the 3D CNN with ConvLSTM, is the way to go for detecting referee signals. The likely reason for this is that the referee signal dataset is relatively small, consisting of only 1409 samples in total. As described in chapter 3, other studies have also shown that networks such as ResNet-18 perform significantly better on small datasets, when pre-training them on larger datasets.

When disregarding the slow increase in accuracy of the 3D CNN model, one can see that the validation accuracy stabilizes at around 50% to 70%, which is still significantly lower than the results achieved by the other two models. This leads to the further conclusion that combining CNNs with RNNs is a more appropriate solution for detecting referee signals than pure CNNs. The same conclusion was reached by the studies proposing the 2D CNN with GRU and 3D CNN with ConvLSTM models for other human gesture datasets (see chapter 3).

As for the implementation on the Nao robot, at this point only the 2D CNN with GRU is able to run on the Nao robot, due to memory issues faced when trying to run the larger 3D CNN with ConvLSTM model. When running the 2D CNN with GRU the robot is able to make real-time predictions of the referee signals every 1.5 seconds, with only a small delay on top of these 1.5 seconds where the robot is collecting the image data. From the limited amount of tests done in the lab, the perceived accuracy is around 90% or slightly above even with people in the background, however the 94.6% achieved on the validation set was not quite reached. Further work needs to be done to determine which sort of backgrounds are still difficult for the model to predict, and the dataset can still be expanded by adding more of such backgrounds to train on.

There is also more work to be done, by testing if the frame rate can be increased from 10 Hz or if the number of frames passed to the network can be decreased from 15. This would let the robot make faster predictions, but needs to be tested rigorously so the robot does not make false predictions too soon for similar looking signals. Also, it needs to be optimized when the robot announces its prediction at the RoboCup challenge, since it has only one chance to do so. Currently a threshold confidence of 90% is set, but more tests need to be made to insure the robot prioritizes announcing a correct prediction over a quick one, since this ensures more points (see challenge description in Appendix C). Finally, if at some point the human referee will be integrated into the regular 5 vs 5 soccer game, the referee detection module needs to be adapted to run in parallel with the rest of the robot's code for the 5 vs 5 soccer game.

Appendix A

Task Description

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CENTER FOR
PROJECT-BASED
LEARNING

Task Description for a Bachelor Thesis on

Visual Referee Detection on Nao Robots for RoboCup SPL 2022

at the Departement of Information Technology and
Electrical Engineering

for

Lukas Molnar
lmolnar@student.ethz.ch

Advisors:	Dr. Seonyeong Heo, seonyeong.heo@pbl.ee.ethz.ch
Professor:	Prof. Dr. Luca Benini, lbenini@iis.ee.ethz.ch
Handout Date:	21.02.2022
Due Date:	30.05.2022

Project Goals

RoboCup is an annual international robotics competition founded in 1996 to promote robotics and artificial intelligence research. The main focus of RoboCup is the soccer game between autonomous robots. Among the five different leagues, RoboCup Standard Platform League (SPL) is for standard humanoid robots, called Nao from Softbank Robotics. Besides the main league, RoboCup SPL also hosts small leagues called technical challenges. The objective of the technical challenges is to solve specific technical problems regarding the gameplay within the league.

One of the technical challenges for RoboCup SPL 2022 is *Visual Referee*. The challenge aims to recognize the signs of the referee wearing red gloves. This project is to design a visual referee detection system for the technical challenge. This project will develop a neural network model for visual referee detection and implement a visual referee detection system for Nao robots. Finally, this project will deploy the model on a robot and evaluate the performance of the system.

Tasks

The project will be split into three phases, as described below:

Phase 1 (Week 1-4)

1. Set up the development environment for the project.
2. Review existing gesture detection models (e.g., [1, 2, 3]).
3. Design a neural network model for visual referee detection from scratch.

Phase 2 (Week 5-11)

1. Collect a referee sign dataset with a robot.
2. Train the neural network model with the dataset.
3. Implement the visual referee detection system.
   a) Implement a visual referee detection module.
   b) Define the behavior of the robot with the module.

Phase 3 (Week 12-14)

1. Evaluate the system in terms of accuracy and latency.
2. Demonstrate the visual referee detection system with a robot.
3. Polish the implementation for submission.
4. Finalize the report and presentation.

Milestones

By the end of **Phase 1** the following should be completed:

- Development environment.
- Neural network design for visual referee detection.

By the end of **Phase 2** the following should be completed:

- Visual referee sign dataset.
- Neural network model trained with the collected dataset.
- First implementation of the visual referee detection system.

By the end of **Phase 3** the following should be completed:

- Final system.
- Final demo with a robot.
- Final report and presentation.

Project Organization

Weekly Report

There will be a weekly report sent by the candidate at the end of every week. The main purpose of this report is to document the project's progress and should be used by the student as a way to communicate any problems that arise during the week.

Project Plan

Within the first month of the project, you will be asked to prepare a project plan. This plan should identify the tasks to be performed during the project and sets deadlines for those tasks. The prepared plan will be a topic of discussion of the first week's meeting between you and your advisers. Note that the project plan should be updated constantly depending on the project's status.

Final Report and Paper

PDF copies of the report are to be turned in. References will be provided by the supervisors by mail and at the meetings during the whole project.

Final Presentation

There will be a 15 min presentation with 5 min Q&A at the end of this project in order to present your results to a wider audience. The exact date will be determined towards the end of the work.

Bibliography

[1] P. Narayana, R. Beveridge, and B. A. Draper, "Gesture recognition: Focus on the hands," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[2] O. Köpüklü, A. Gunduz, N. Kose, and G. Rigoll, "Real-time hand gesture detection and classification using convolutional neural networks," in *2019 14th IEEE International Conference on Automatic Face Gesture Recognition (FG 2019)*, 2019, pp. 1–8.

[3] L. Zhang, G. Zhu, L. Mei, P. Shen, S. A. A. Shah, and M. Bennamoun, "Attention in convolutional lstm for gesture recognition," in *Advances in Neural Information Processing Systems*, S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, Eds., vol. 31. Curran Associates, Inc., 2018. [Online]. Available: https://proceedings.neurips.cc/paper/2018/file/287e03db1d99e0ec2edb90d079e142f3-Paper.pdf

Appendix B

Declaration of Originality

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Visual Referee Detection on Nao Robots for RoboCup SPL 2022

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**
Molnar	Lukas

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
Zurich, 30.05.2022	[illegible]

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.

Appendix C

RobCup Visual Referee Challenge Description

B.2 Visual Referee Challenge

B.2.1 Challenge Goal

In the current SPL rules, the only time that a robot is required to listen directly to the human referees is for the kick-off whistle. Otherwise, all human referee decisions are communicated to the robots via electronic GameController messages. In moving towards the 2050 RoboCup goal, robots will need to directly interpret referee calls and signals (such as whistles, spoken calls and hand signals), rather than receive information from an external electronic source.

This technical challenge tests a robot's ability to identify three categories of hand signals:

1. Static hand signals with one hand.
2. Static hand signals with two hands.
3. Dynamic (motion) hand signals with a one or two hands.

The intent of this challenge is to choose a *subset* of potential referee calls in SPL games and test ability of a robot to recognise different types of hand signals in preparation for adoption in RoboCup games, rather than compile a complete set of all referee signals.

B.2.2 Challenge setup

One robot of the challenged team is placed in the center circle facing the head referee, standing upright (stiffened) with both hands by its side. The robot should be running in the challenge mode at the start of the challenge. The team is free to choose how the software is started.

The head referee must stand on the T-junction of the center field line, opposite to the GC. The referee must wear a "black-and-white" stripped referee jersey (if available) and must wear red gloves. The purpose of this clothing is to clearly distinguish the referee and their hands from background people.

The description of this challenge (and hand-signals) is described based on the viewpoint of the head referee. In these descriptions the "red team" is defined as playing from left-to-right, and the "blue team" as playing from right-to-left. The use of colours for identifying teams is used to give equivalence to the head referee calls during SPL games.

The general procedure is as follows:

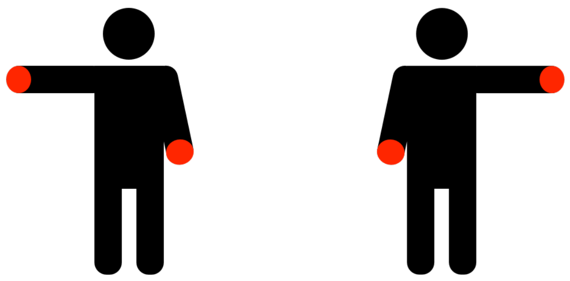
1. The referee selects a hand-signal (and direction where applicable).
2. The referee blows the whistle once. The referee may use one hand to hold (and blow) the whistle. The referee may instead leave the whistle held in their mouth without their hands.
3. 1 s after blowing the whistle, the referee indicates the hand-signal for 10 s.
4. During that time or within an additional 10 s, the robot must indicate the hand-signal that it has identified by:
   (a) Using its hand(s) to *mirror* the referee's signal. That is, if the referee given a signals for a decision for the "red team" using their right hand, the robot should *mirror* this signal using the robot's *left hand*, as the robot is facing the referee.
   (b) Providing an audio phrase the referee's decision, e.g., "'Goal Red Team"'. The exact wording is up to the team, but should clearly identify the referee's signal.
   (c) (Optional), At the discretion of the GC/TCM developers, TCP messages may also be sent and interpreted to display the outcome of the robot's decision, however, this is not a requirement for the execution of the challenge.
5. The length of time taken for the robot to indicate it's interpretation from the referee blowing the whistle is measured (rounded-up to the nearest second).
6. If the robot cannot identify the signal, it should remain motionless and provide no audio output. A robot may continue to pose in the same position as the identified last signal.
7. While not providing a hand-signal or using the whistle, the head referee must keep both hands flat and motionless by their side.

This procedure will be repeated *five* times. The referee should choose 2 one-hand static signals, 2 two-hand static signals, and 1 dynamic signal. Within each type, the referee randomly chooses a hand-signal and direction.The same hand-signal may be chosen twice (with different directions).

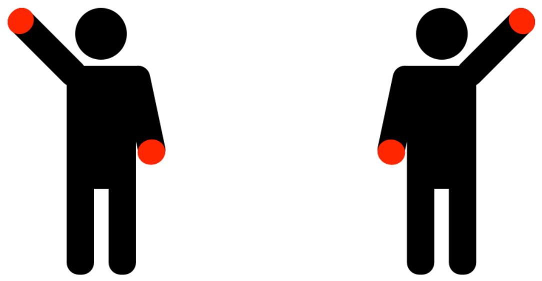
B.2.3 Available Hand-Signals

Each hand-signal for the challenge is described and pictured. Note that for the purpose of clarity, these do not necessarily correspond to human soccer hand-signals.

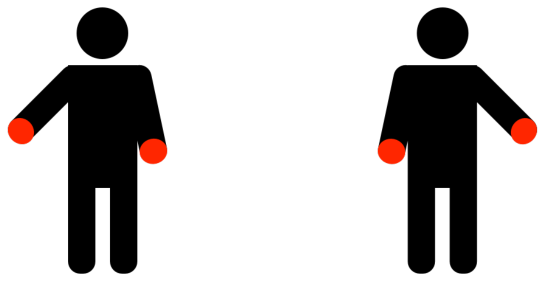
- **Kick-in ⟨colour⟩ Team.** One-handed signal. One arm, extended horizontally in the direction of the half of the field corresponding to the team that receives the Kick-in Free Kick. That is, right arm extended for the "Blue team", and left arm extended for the "Red team".



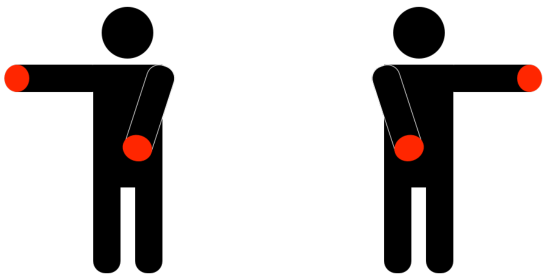
- **Goal Kick ⟨colour⟩ Team.** One-handed signal. One arm, extended 45-degree *up* in the direction of the end of the field where the goal kick will occur. That is, right arm extended for the "Blue team", and left arm extended for the "Red team".



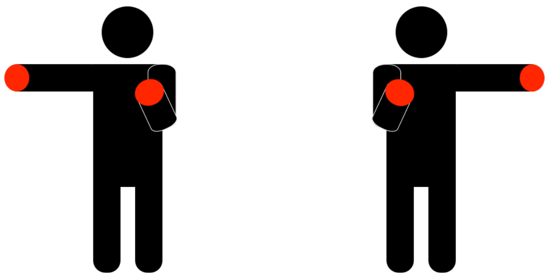
- **Corner Kick ⟨colour⟩ Team.** One-handed signal. One arm, extended 45-degree *down* in the direction of the end of the field where the corner kick will occur. That is, right arm extended for the "Red team", and left arm extended for the "Blue team".



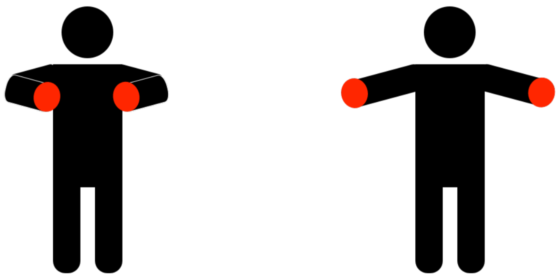
- **Goal ⟨colour⟩ Team.** Two-handed signal. One arm, extended pointing at the center circle. One arm, extended horizontally in the direction of the half of the field corresponding to the team that scored the goal. That is, right arm extended for the "Blue team", and left arm extended for the "Red team".



- **Pushing Free-kick ⟨colour⟩ Team.** Two-handed signal. One arm, vertical with bent elbow and palm facing in the direction of the extended arm. One arm, extended horizontally in the in the direction of the half of the field corresponding to the team that is *given* the penalty. That is, right arm extended for the "Blue team", and left arm extended for the "Red team".



- **Full-Time.** Dynamic two-handed signal. Both arms slowly move symmetrically inward and outwards on a horizontal plane, bending at the elbows. Note, for the purpose of this challenge, the whistle associated with this signal should be a *single* blow, unlike in normal SPL games.



B.2.4 Challenge evaluation

A team scores 1 point for every hand-signal that is correctly identified. A team score an additional 1 point for correctly identifying the team corresponding to the signal (where appropriate). A team looses 1 point for incorrectly identifying a hand-signal (note a team may have a negative final score). The total time for the robot to identify each hand-signal is summed (If a robot fails to identify a hand-signal the time for the hand-signal is $10\,\mathrm{s}$. If a robot incorrectly identifies a hand-signal, the time is how long the robot took to provide the incorrect identification).

Teams are ranked by their total points. In the event of tie-breaks, the team with the fastest total time to identify all hand-signals is ranked higher. The team with the highest total points, and lowest total time (for tie-breaks), wins the challenge.

Appendix D

Model Architectures

Following are the lists of all network layers for the 3D CNN, as well as 2D CNN with GRU architectures. The input shape of all networks is: (None, 15, 112, 112, 3). "None" is a placeholder for the batch size.

Layer (type)	Output Shape	Parameters
conv3d_1 (Conv3D)	(None, 15, 112, 112, 32)	2,624
batch_norm_1 (BatchNormalization)	(None, 15, 112, 112, 32)	128
max_pooling3d_1 (MaxPooling3D)	(None, 7, 56, 56, 64)	0
conv3d_2 (Conv3D)	(None, 7, 56, 56, 64)	55,360
batch_norm_2 (BatchNormalization)	(None, 7, 56, 56, 64)	256
max_pooling3d_2 (MaxPooling3D)	(None, 3, 28, 28, 64)	0
conv3d_3 (Conv3D)	(None, 3, 28, 28, 64)	221,312
batch_norm_3 (BatchNormalization)	(None, 3, 28, 28, 64)	512
max_pooling3d_3 (MaxPooling3D)	(None, 1, 14, 14, 128)	0
conv3d_4 (Conv3D)	(None, 1, 14, 14, 128)	442,496
batch_norm_4 (BatchNormalization)	(None, 1, 14, 14, 128)	512
max_pooling3d_4 (MaxPooling3D)	(None, 1, 7, 7, 128)	0
flatten (Flatten)	(None, 6272)	0
dense_1 (Dense)	(None, 256)	1,605,888
batch_norm_5 (BatchNormalization)	(None, 256)	1024
dropout_1 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 256)	65,792
batch_norm_6 (BatchNormalization)	(None, 256)	1024
dropout_2 (Dropout)	(None, 256)	0
dense_3 (Dense)	(None, 64)	16,448
dense_4 (Dense)	(None, 12)	780
Total parameters: 2,414,156		
Trainable parameters: 2,412,428		
Non-trainable parameters: 1,728		

Table D.1.: 3D CNN Architecture

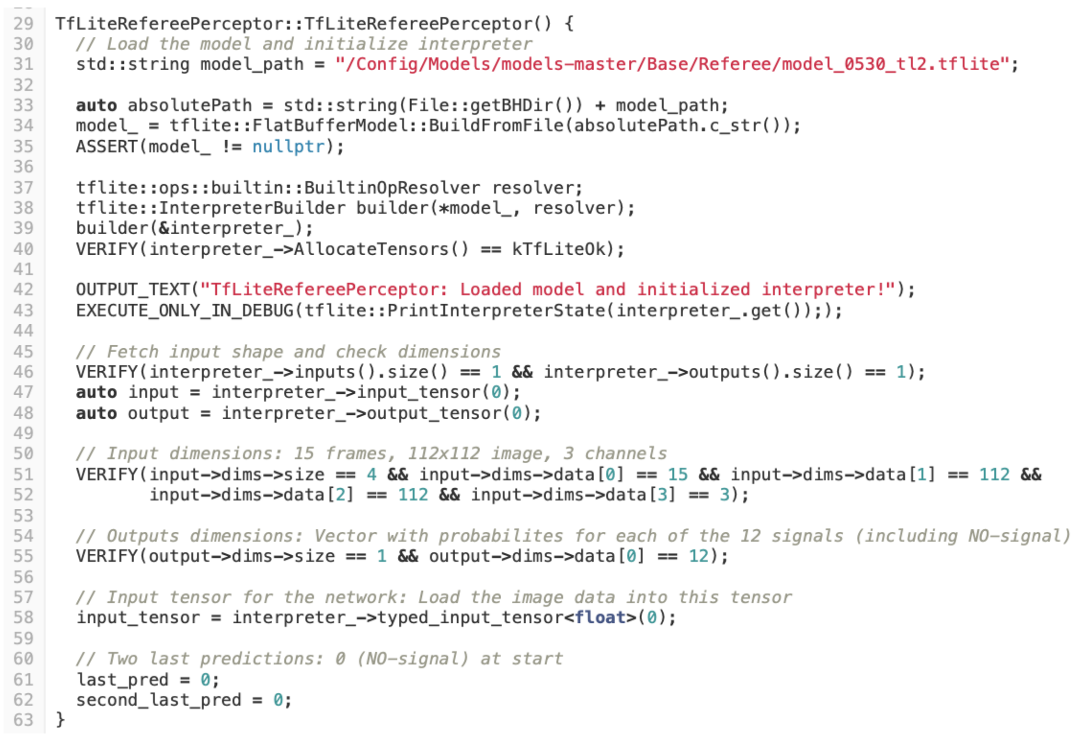
Layer (type)	Output Shape	Parameters
td_mobilenet (TimeDistributed MobileNet)	(None, 15, 3, 3, 1024)	3,228,864
td_bnorm (TimeDistributed BatchNormalization)	(None, 15, 3, 3, 1024)	4096
td_maxpool (TimeDistributed MaxPooling2D)	(None, 15, 1, 1, 1024)	0
td_flatten (TimeDistributed Flatten)	(None, 15, 1024)	0
gru (GRU)	(None, 64)	209,280
dropout_1 (Dropout)	(None, 64)	0
dense_1 (Dense)	(None, 128)	8,320
dropout_2 (Dropout)	(None, 128)	0
dense_2 (Dense)	(None, 12)	1,548
Total parameters: 3,452,108		
Trainable parameters: 3,428,172		
Non-trainable parameters: 23,936		

Table D.2.: 2D CNN with GRU Architecture

Appendix E

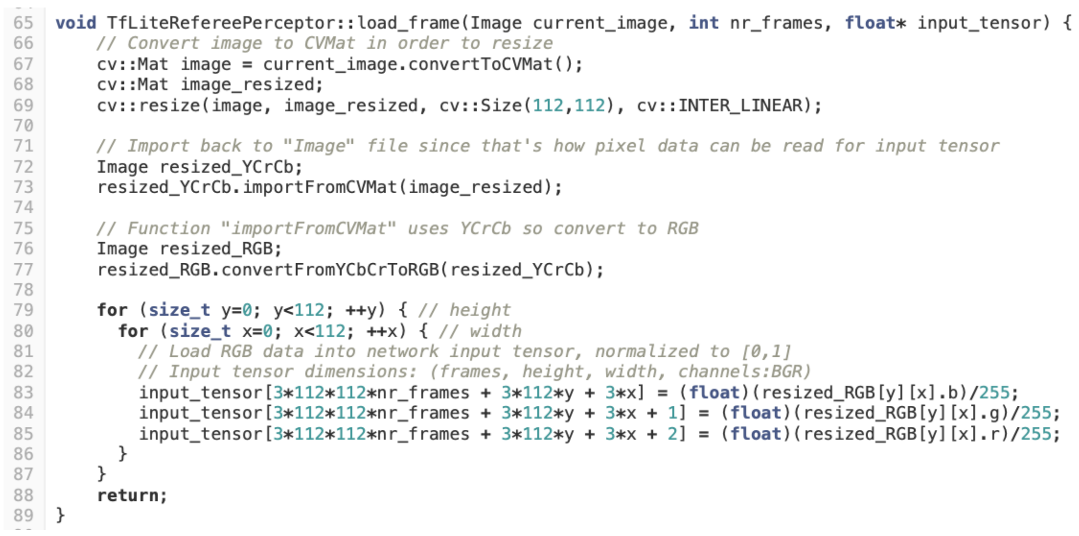
Nao Robot Code

On the following three pages the code is presented for the `TfLiteRefereePerceptor` module that runs on the Nao robot. The explanation of the code can be found in section 5.3.



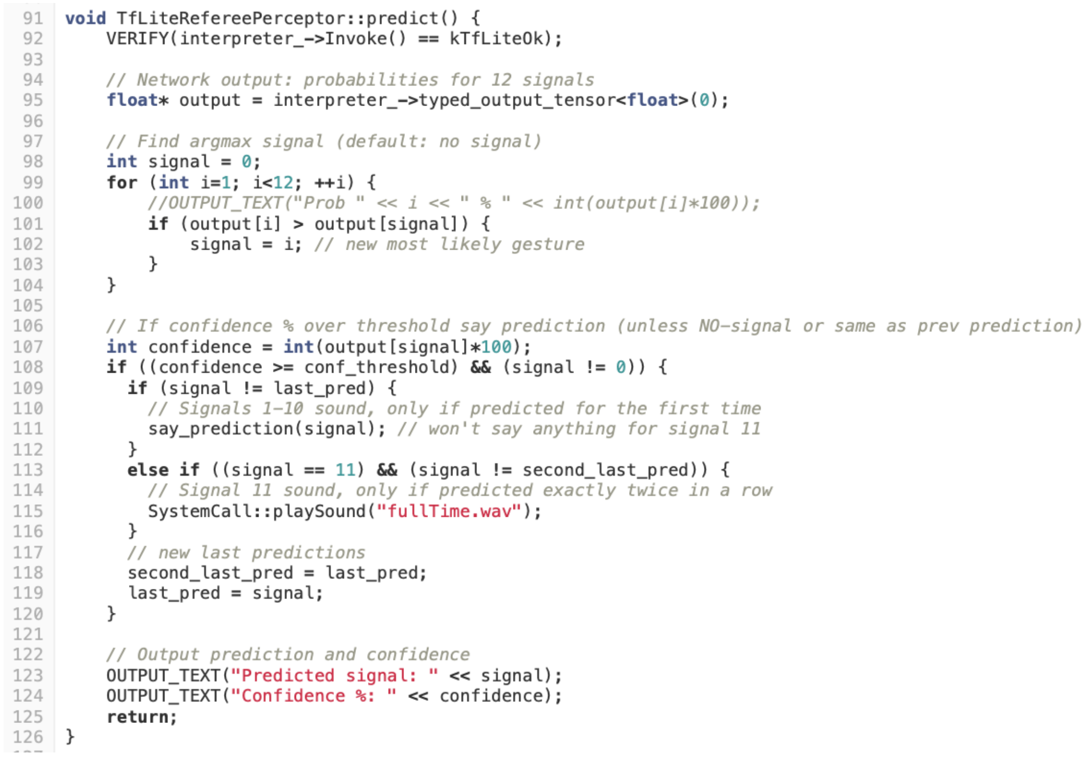
```cpp
TfLiteRefereePerceptor::TfLiteRefereePerceptor() {
  // Load the model and initialize interpreter
  std::string model_path = "/Config/Models/models-master/Base/Referee/model_0530_tl2.tflite";

  auto absolutePath = std::string(File::getBHDir()) + model_path;
  model_ = tflite::FlatBufferModel::BuildFromFile(absolutePath.c_str());
  ASSERT(model_ != nullptr);

  tflite::ops::builtin::BuiltinOpResolver resolver;
  tflite::InterpreterBuilder builder(*model_, resolver);
  builder(&interpreter_);
  VERIFY(interpreter_->AllocateTensors() == kTfLiteOk);

  OUTPUT_TEXT("TfLiteRefereePerceptor: Loaded model and initialized interpreter!");
  EXECUTE_ONLY_IN_DEBUG(tflite::PrintInterpreterState(interpreter_.get()););

  // Fetch input shape and check dimensions
  VERIFY(interpreter_->inputs().size() == 1 && interpreter_->outputs().size() == 1);
  auto input = interpreter_->input_tensor(0);
  auto output = interpreter_->output_tensor(0);

  // Input dimensions: 15 frames, 112x112 image, 3 channels
  VERIFY(input->dims->size == 4 && input->dims->data[0] == 15 && input->dims->data[1] == 112 &&
         input->dims->data[2] == 112 && input->dims->data[3] == 3);

  // Outputs dimensions: Vector with probabilites for each of the 12 signals (including NO-signal)
  VERIFY(output->dims->size == 1 && output->dims->data[0] == 12);

  // Input tensor for the network: Load the image data into this tensor
  input_tensor = interpreter_->typed_input_tensor<float>(0);

  // Two last predictions: 0 (NO-signal) at start
  last_pred = 0;
  second_last_pred = 0;
}
```

Figure E.1.: Module Constructor



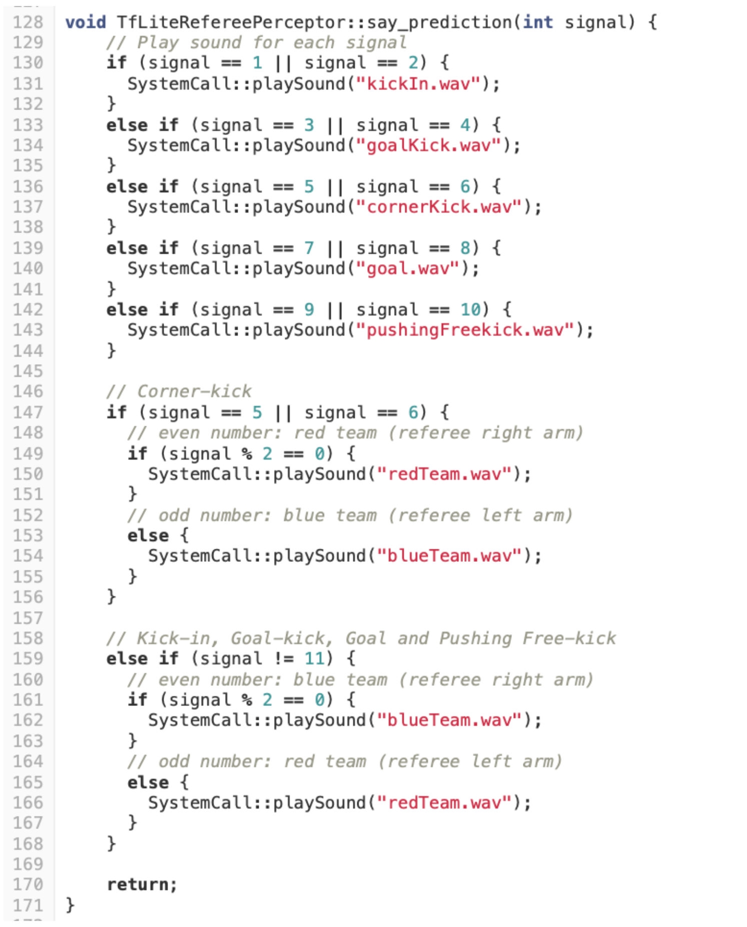
```cpp
void TfLiteRefereePerceptor::load_frame(Image current_image, int nr_frames, float* input_tensor) {
    // Convert image to CVMat in order to resize
    cv::Mat image = current_image.convertToCVMat();
    cv::Mat image_resized;
    cv::resize(image, image_resized, cv::Size(112,112), cv::INTER_LINEAR);

    // Import back to "Image" file since that's how pixel data can be read for input tensor
    Image resized_YCrCb;
    resized_YCrCb.importFromCVMat(image_resized);

    // Function "importFromCVMat" uses YCrCb so convert to RGB
    Image resized_RGB;
    resized_RGB.convertFromYCbCrToRGB(resized_YCrCb);

    for (size_t y=0; y<112; ++y) { // height
      for (size_t x=0; x<112; ++x) { // width
        // Load RGB data into network input tensor, normalized to [0,1]
        // Input tensor dimensions: (frames, height, width, channels:BGR)
        input_tensor[3*112*112*nr_frames + 3*112*y + 3*x] = (float)(resized_RGB[y][x].b)/255;
        input_tensor[3*112*112*nr_frames + 3*112*y + 3*x + 1] = (float)(resized_RGB[y][x].g)/255;
        input_tensor[3*112*112*nr_frames + 3*112*y + 3*x + 2] = (float)(resized_RGB[y][x].r)/255;
      }
    }
    return;
}
```

Figure E.2.: Module Function `load_frame`



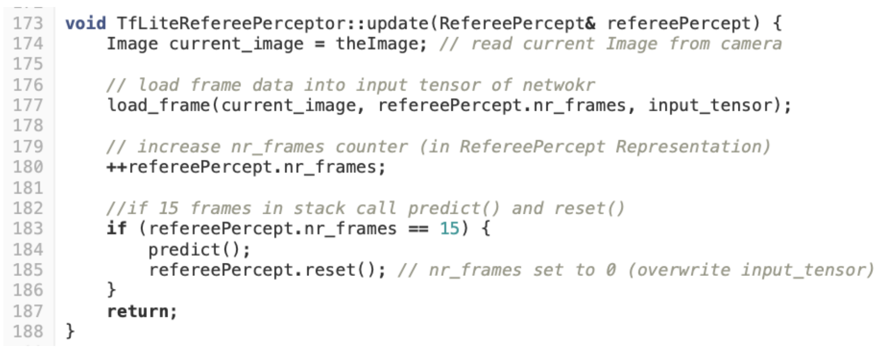
```cpp
void TfLiteRefereePerceptor::predict() {
    VERIFY(interpreter_->Invoke() == kTfLiteOk);

    // Network output: probabilities for 12 signals
    float* output = interpreter_->typed_output_tensor<float>(0);

    // Find argmax signal (default: no signal)
    int signal = 0;
    for (int i=1; i<12; ++i) {
        //OUTPUT_TEXT("Prob " << i << " % " << int(output[i]*100));
        if (output[i] > output[signal]) {
            signal = i; // new most likely gesture
        }
    }

    // If confidence % over threshold say prediction (unless NO-signal or same as prev prediction)
    int confidence = int(output[signal]*100);
    if ((confidence >= conf_threshold) && (signal != 0)) {
      if (signal != last_pred) {
        // Signals 1-10 sound, only if predicted for the first time
        say_prediction(signal); // won't say anything for signal 11
      }
      else if ((signal == 11) && (signal != second_last_pred)) {
        // Signal 11 sound, only if predicted exactly twice in a row
        SystemCall::playSound("fullTime.wav");
      }
      // new last predictions
      second_last_pred = last_pred;
      last_pred = signal;
    }

    // Output prediction and confidence
    OUTPUT_TEXT("Predicted signal: " << signal);
    OUTPUT_TEXT("Confidence %: " << confidence);
    return;
}
```

Figure E.3.: Module Function `predict`



```
void TfLiteRefereePerceptor::say_prediction(int signal) {
    // Play sound for each signal
    if (signal == 1 || signal == 2) {
      SystemCall::playSound("kickIn.wav");
    }
    else if (signal == 3 || signal == 4) {
      SystemCall::playSound("goalKick.wav");
    }
    else if (signal == 5 || signal == 6) {
      SystemCall::playSound("cornerKick.wav");
    }
    else if (signal == 7 || signal == 8) {
      SystemCall::playSound("goal.wav");
    }
    else if (signal == 9 || signal == 10) {
      SystemCall::playSound("pushingFreekick.wav");
    }

    // Corner-kick
    if (signal == 5 || signal == 6) {
      // even number: red team (referee right arm)
      if (signal % 2 == 0) {
        SystemCall::playSound("redTeam.wav");
      }
      // odd number: blue team (referee left arm)
      else {
        SystemCall::playSound("blueTeam.wav");
      }
    }

    // Kick-in, Goal-kick, Goal and Pushing Free-kick
    else if (signal != 11) {
      // even number: blue team (referee right arm)
      if (signal % 2 == 0) {
        SystemCall::playSound("blueTeam.wav");
      }
      // odd number: red team (referee left arm)
      else {
        SystemCall::playSound("redTeam.wav");
      }
    }

    return;
}
```

Figure E.4.: Module Function `say_prediction`



```
void TfLiteRefereePerceptor::update(RefereePercept& refereePercept) {
    Image current_image = theImage; // read current Image from camera

    // load frame data into input tensor of netwokr
    load_frame(current_image, refereePercept.nr_frames, input_tensor);

    // increase nr_frames counter (in RefereePercept Representation)
    ++refereePercept.nr_frames;

    //if 15 frames in stack call predict() and reset()
    if (refereePercept.nr_frames == 15) {
        predict();
        refereePercept.reset(); // nr_frames set to 0 (overwrite input_tensor)
    }
    return;
}
```

Figure E.5.: Module Function `update`

Appendix F

File Structure

The repository is not made public, due to the fact that the RoboCup competition has not taken place yet. For access to the repository contact: lukas.molnar@bluewin.ch.

The file structure of the repository can be found on the following page.

```
/
├── README ........ A README with instructions for using the dataset and models
├── main.py ......................................... Main file to train models
├── model_builder.py ............ Builds models and prepares dataset for training
├── models_cnn_clstm.py .. Models not covered in this report (worse performance)
├── models_cnn_tl.py .............3D CNN model and 2D CNN with GRU model
├── models_res3d_clstm.py .....................3D CNN with ConvLSTM model
├── figures/ ................................. Folder for confusion matrix figures
│   ├── evaluate.py ......... Creates confusion matrix figure (enter path to model)
│   └── ...(Confusion matrices)
├── histories/ .............................Folder for histories of trained models
│   ├── create_plots.py ............................ Creates plot of model history
│   └── ...(History files)
├── outputs/ ........................... Folder for model outputs during training
│   └── ...(Output files)
├── pretrained/ ..................... Folder for pretrained model files (on IsoGD)
│   └── ...(Pretrained ".h5" files)
├── referee_dataset/ .............................Folder for referee dataset files
│   ├── create_labels.py ...........Create CSV file of labels once frames are split
│   ├── edit_frames.py  Split data into sequences length 15 and replace greenscreen
│   ├── edit_frames_no_greenscreen.py .......Split data into sequences length 15
│   ├── split_train_val.py ............Split processed data into train and val sets
│   └── ...(Background images)
└── tflite_files/ .......................Folder for TfLite files of trained models
    ├── tflite_converter.py ..................Convert ".h5" model file to TfLite
    ├── tflite_converter_quant.py ......... Convert to TfLite using quantization
    └── ...(TfLite files)
```

Bibliography

[1] Pérez-Enciso and L. Zingaretti, “A guide for using deep learning for complex trait genomic prediction,” *Genes*, vol. 10, p. 553, 07 2019.

[2] R. Wattenhofer, “Computational thinking,” *Distributed Computing, ETH Zürich*, 2021.

[3] DProgrammer, “Rnn, lstm & gru,” accessed on 29.05.2022. [Online]. Available: http://dprogrammer.org/rnn-lstm-gru

[4] K. Hara, H. Kataoka, and Y. Satoh, “Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 6546–6555.

[5] L. Zhang, G. Zhu, L. Mei, P. Shen, S. A. A. Shah, and M. Bennamoun, “Attention in convolutional lstm for gesture recognition,” in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, ser. NIPS’18. Red Hook, NY, USA: Curran Associates Inc., 2018, p. 1957–1966.

[6] “Robocup,” accessed on 29.05.2022. [Online]. Available: https://www.robocup.org

[7] “Robocup standard platform league,” accessed on 29.05.2022. [Online]. Available: https://spl.robocup.org

[8] S. Robotics, “Nao,” accessed on 29.05.2022. [Online]. Available: https://www.softbankrobotics.com/emea/en/nao

[9] R. T. Committee, “Robocup standard platform league (nao) rule book,” accessed on 29.05.2022. [Online]. Available: https://spl.robocup.org/wp-content/uploads/SPL-Rules-master.pdf

[10] D. K. Singh, “3d-cnn based dynamic gesture recognition for indian sign language modeling,” *Procedia Computer Science*, vol. 189, pp. 76–83, 2021, aI in Computational Linguistics. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1877050921011650

[11] S. Ioffe and C. Szegedy, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," 2015. [Online]. Available: https://arxiv.org/abs/1502.03167

[12] Keras, "Batchnormalization layer," accessed on 29.05.2022. [Online]. Available: https://keras.io/api/layers/normalization_layers/batch_normalization/

[13] V. Lendave, "Lstm vs gru in recurrent neural network: A comparative study," 2021, accessed on 29.05.2022. [Online]. Available: https://analyticsindiamag.com/lstm-vs-gru-in-recurrent-neural-network-a-comparative-study/

[14] Peltarion, "Categorical crossentropy," accessed on 29.05.2022. [Online]. Available: https://peltarion.com/knowledge-center/documentation/modeling-view/build-an-ai-model/loss-functions/categorical-crossentropy

[15] U. of Central Florida, "Ucf101 - action recognition dataset," accessed on 29.05.2022. [Online]. Available: https://www.crcv.ucf.edu/data/UCF101.php

[16] S. Lab, "Hmdb: a large human motion database," accessed on 29.05.2022. [Online]. Available: https://serre-lab.clps.brown.edu/resource/hmdb-a-large-human-motion-database/

[17] D. Kushwaha, "Gesture recognition using conv3d and transfer learning convolutional rnn architecture," accessed on 29.05.2022. [Online]. Available: https://github.com/deepakush/Gesture-Recognition-Using-Conv3D-and-transfer-learning-Convolutional-RNN-architecture

[18] TensorFlow, "tf.keras.applications.mobilenet.mobilenet," accessed on 29.05.2022. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/applications/mobilenet/MobileNet

[19] L. Zhang, G. Zhu, P. Shen, J. Song, S. A. Shah, and M. Bennamoun, "Learning spatiotemporal features using 3dcnn and convolutional lstm for gesture recognition," in *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, 2017, pp. 3120–3128.

[20] J. Wan, "Chalearn lap isogd database," accessed on 29.05.2022. [Online]. Available: http://www.cbsr.ia.ac.cn/users/jwan/database/isogd.html

[21] G. Zhu, "Attentionconvlstm," accessed on 29.05.2022. [Online]. Available: https://github.com/GuangmingZhu/AttentionConvLSTM

[22] TensorFlow, "tf.keras.layers.convlstm2d," accessed on 29.05.2022. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/ConvLSTM2D

[23] GeeksforGeeks, "Intuition of adam optimizer," accessed on 29.05.2022. [Online]. Available: https://www.geeksforgeeks.org/intuition-of-adam-optimizer/

[24] Viso.ai, "Tensorflow lite - real-time computer vision on edge devices," accessed on 29.05.2022. [Online]. Available: https://viso.ai/edge-ai/tensorflow-lite/

[25] TensorFlow, "Tensorflow lite c++ api reference," accessed on 29.05.2022. [Online]. Available: https://www.tensorflow.org/lite/api_docs/cc

[26] OpenCV, "Image file reading and writing," accessed on 29.05.2022. [Online]. Available: https://docs.opencv.org/3.4/d4/da8/group__imgcodecs.html

[27] V. Sharma, M. Jaiswal, A. Sharma, S. Saini, and R. Tomar, "Dynamic two hand gesture recognition using cnn-lstm based networks," in *2021 IEEE International Symposium on Smart Electronic Systems (iSES)*, 2021, pp. 224–229.