CHAPTER 1
INTRODUCTION

AFFECTIVE COMPUTING

Affective computing (sometimes called artificial emotional intelligence, or emotion AI) is the study and development of systems and devices that can recognize, interpret, process and simulate human effects. It is an interdisciplinary field spanning computer science, psychology and cognitive science. Detecting emotional information begins with passive sensors which capture data about the user's physical state or behavior without interpreting the input. The data gathered is analogous to the cues humans use to perceive emotions in others. For example, a video camera might capture facial expressions, body posture and gestures, while a microphone might capture speech. Other sensors detect emotional cues by directly measuring physiological data, such as skin temperature and galvanic resistance. The detection and processing of facial expression are achieved through various methods such as optical flow, hidden Markov models, neural network processing or active appearance models. More than one modalities can be combined or fused (multimodal recognition, e.g. facial expressions and speech prosody, facial expressions and hand gestures, or facial expressions with speech and text for multimodal data and metadata analysis) to provide a more robust estimation of the subject's emotional state.

1.1. SYSTEM OVERVIEW

This System primarily involves building a method of enabling the tracking of facial expressions in ASD affected children and analyzing the affect and engagement levels of the children in socio-communication interactions by the video feed acquired in real time. In this system, the face of the ASD affected children is segmented from the video feed using faster R-CNN and the facial expressions from the images are detected by Resnet and their engagement level is estimated with the help of the facial expressions detected from the video feed. The model is trained with multi culture data where this is the training and validation data allotted and unknown images are given for testing. These facial expressions are segmented and learnt during the training by faster R-CNN and the live video feed detects the current engagement level of ASD affected children.

1.2. SCOPE OF THE PROJECT

The recent breakthroughs in Image processing produced many fields in deep learning. Affective computing is one area which has improved over years by image processing and deep learning. There are many papers developed over these years in affective computing but still there are many children been affected from autism. Autism Spectrum Disorder is a serious problem occurring in children affecting their social life and there is no fully backed data-driven approach for Autism Therapy. ASD affected children possess a typical facial expressions compared to normal children. Detection of their facial expressions will help to understand their feelings and engagement levels which is the main solution developed in affective computing. This will help to detect their level of improvement know whether they are gaining the abilities of a normal person.

CHAPTER 2

LITERATURE SURVEY

[1]CultureNet: A Deep Learning Approach for Engagement Intensity Estimation from Face Images of Children with Autism

AUTHOR: Ognjen Rudovic, Yuria Utsumi, Jaeryoung Lee, Javier Hernandez, Eduardo Castello Ferrer, Bjorn Schuller, Rosalind W. Picard

YEAR: 2018

Performance of deep learning models in the task of automated engagement estimation from face images of children with autism was introduced. Specifically, video data of 30 children with different cultural backgrounds (Asia vs. Europe) recorded during single session of robot-assisted autism therapy was used. Thorough evaluation of the proposed deep architectures for the target task, including within- and across-culture evaluations, as well as when using the child-independent and child-dependent settings was performed. Novel deep learning model, named CultureNet, which efficiently leverages the multi-cultural data when performing the adaptation of the proposed deep architecture to the target culture and child was introduced to detect the facial expressions

[2] The Affective Computing Approach to Affect Measurement

AUTHOR: Sidney D'Mello, Arvid Kappas, Jonathan Gratch

YEAR: 2017

Affective computing approach towards automated affect measures that jointly model machine-readable physiological/behavioral signals with affect estimates as reported by humans or experimentally elicited. The conceptual and computational foundations of the approach followed by two case studies were described: one on discrimination between genuine and faked expressions of pain in

the lab and second on measuring nonbasic affect in the wild. Applications of the measures, analyze measurement accuracy and generalizability and highlight advances afforded by computational tipping points, such as big data, wearable sensing, crowdsourcing and deep learning were discussed.

[3] Multi-layer affective computing model based on emotional psychology

AUTHOR: Qingyuan Zhou

YEAR: 2017

The factors and transforms of affective state were analyzed based on affective psychology theory. After that, multi-layer affective decision model was proposed by establishing mapping relation among character, mood and motion. The model reflected the changes of mood and emotion spaces based on different characters. Experiment showed that human emotion characteristics accorded with theory and law, thus providing reference for modeling of human–computer interaction system.

[4]Multi-modal Approach for Affective Computing

AUTHOR: Siddharth, Tzyy-Ping Jung, Terrence J. Sejnowski

YEAR: 2018

Using multi-modal AMIGOS dataset, performance of human emotion classification using multiple computational approaches applied to face videos and various bio-sensing modalities. Using a novel method for compensating physiological baseline an increase in the classification accuracy of various approaches was shown. Finally, a multi-modal emotion-classification approach in the domain of affective computing research was introduced.

[5]Going Deeper in Facial Expression Recognition using Deep Neural Networks

AUTHOR: Ali Mollahosseini, David Chan, Mohammad H. Mahoor

YEAR: 2016

A deep neural network architecture to address the FER problem across multiple well known standard face datasets. Specifically, this network has two convolutional layers each followed by max pooling and then four Inception layers. Proposed network was a single component architecture that took registered facial images as the input and classified them into either of the six basic or the neutral expressions. Comprehensive experiments on seven publicly available facial expression databases, viz. MultiPIE, MMI, CK+, DISFA, FERA, SFEW and FER2013 were conducted. The results of this system architecture were comparable to or better than the state-of-the-art methods and better than traditional convolutional neural networks in both accuracy and training time.

[6] Perceptual and affective mechanisms in facial expression recognition: An integrative review

AUTHOR: Manuel G. Calvo, Lauri Nummenmaa

YEAR: 2015

Role of visual and emotional factors in expression recognition reached three major conclusions. First, behavioral, neurophysiological and computational measures indicated that basic expressions were reliably recognized and discriminated from one another, albeit the effect may be inflated by the use of prototypical expression stimuli and forced-choice responses. Second, affective content along the dimensions of valence and arousal was extracted early from facial expressions, although this coarse affective representation contributed minimally to categorical recognition of specific expressions. Third, the physical

configuration and visual saliency of facial features contributed significantly to expression recognition, with “emotionless” computational models being able to reproduce some of the basic phenomena demonstrated in human observers

[7]Facial expression recognition from video sequences: temporal and static modeling

AUTHOR: Ira Cohen, Nicu Sebe, Ashutosh Garg, Lawrence S. Chen, Thomas S. Huang

YEAR: 2003

Bayesian network classifiers for classifying expressions from video, focusing on changes in distribution assumptions and feature dependency structures. In particular Naive–Bayes classifiers, distribution from Gaussian to Cauchy, Gaussian Tree-Augmented Naive Bayes (TAN) classifiers learnt the dependencies among different facial motion features. Facial expression recognition from live video input using temporal cues was introduced. Existing methods were exploited and a new architecture of hidden Markov models (HMMs) for automatically segmenting and recognizing human facial expression from video sequences was introduced. The architecture performs both segmentation and recognition of the facial expressions automatically using a multi-level architecture composed of an HMM layer and a Markov model layer.

[8]Facial Emotion Recognition Based on Biorthogonal Wavelet Entropy, Fuzzy Support Vector Machine and Stratified Cross Validation

AUTHOR: Yu Dong Zhang et al.

YEAR: 2016

A new emotion recognition system based on facial expression images which enrolled 20 subjects and each subject posed seven different emotions: happy, sadness, surprise, anger, disgust, fear and neutral are annotated. Afterward, a biorthogonal wavelet entropy to extract multiscale features was employed and fuzzy multiclass support vector machine were used to be the classifier. The stratified cross validation was employed as a strict validation model. The statistical analysis showed, method achieved an overall accuracy of 96.77±0.10% which is superior to three state-of-the-art methods.

[9] Engagement Analysis Through Computer Vision

AUTHOR: Zachary M MacHardy, Kenneth Syharath, Prasun Dewan

YEAR: 2012

Real time feedback System to online presenter about the engagement level of the audience where facial recognition by computer vision ran in background. The tool made inferences by using computer vision and machine learning techniques to analyze the faces of audience members.

[10] Crowdsourcing Facial Responses to Online Videos

AUTHOR: Daniel McDuff, Rana el Kaliouby, Rosalind W. Picard

YEAR: 2015

Enabled 3,268 trackable face videos to be collected, analyzed in under two months. Each participant viewed one or more commercials while their facial response was recorded and analyzed. Data showed significantly different intensity and dynamics patterns of smile responses between subgroups who reported liking the commercials versus those who did not and collected over three million videos of facial responses in over 75 countries using same methodology, enabling facial analytics to become significantly more accurate and validated across five continents. Many new insights have been discovered based on crowdsourced facial data, enabling Internet-based measurement of facial responses to become reliable and proven.

CHAPTER 3
SYSTEM ANALYSIS

3.1. EXISTING SYSTEM

There are many Video-on-Demand platforms where the autism spectrum affected children are monitored and insights given. There are many picture based and text based communication platforms (like avaz app) where they provide specific activities to the autism children. The current system the features from these optimized models can be extracted by blinded non expert raters from 3-minute home videos of children with and without ASD to arrive at a rapid and accurate machine learning autism classification.

3.1.1. DISADVANTAGES OF EXISTING SYSTEM

Traditional and Time-consuming Approach:

The standard approaches to diagnosing autism spectrum disorder (ASD) evaluate between 20 and 100 behaviors and take several hours to complete for evaluating the common behavioral traits.

Real time lagging of video data:

These results support the hypothesis that feature tagging of home videos for machine learning classification of autism cannot yield accurate outcomes in long time video frames, using mobile devices.

3.2. PROPOSED SYSTEM

The proposed system includes a Video-on-demand platform where the system is backed with state-of-the-art machine learning where the facial expressions are detected which can lead to estimate the engagement level. The video feed is acquired in real time from the webcam and the video feed is constantly fed into the deep learning system. The images are being segmented from the real time feed by faster R-CNN and the facial expressions are extracted from resnet and the engagement estimation is measured from the facial expressions which is trained for finding the observed affective states such as valence (pleasure-displeasure continuum) and arousal(alertness) and engagement in the task to each child.

3.2.1. ADVANTAGES OF PROPOSED SYSTEM

Facial Expressions Detection-

This proposed system involves detecting the facial expressions of the autism affected children by faster R-CNN and resnet which hard to interpret where they possess atypical facial characteristics with respect to their nervous system.

Estimation of Engagement level-

The engagement levels of the autism children is estimated by categorizing the facial expressions in real time video feed.

3.3. REQUIREMENTS SPECIFICATION

3.3.1. HARDWARE REQUIREMENTS

- Operating system: Windows 7 or newer, 64-bit mac OS 10.9+, or Linux.
- System architecture: 64-bit x86, 32-bit x86 with Windows or Linux.
- CPU: Intel Core 2 Quad CPU Q6600 @ 2.40GHz or greater.
- RAM: 8 Gigabyte or greater
- GPU: GTX 1080 Ti. Specs. VRAM: 11 GB. Memory bandwidth: 484 GBs/second.

3.3.2. SOFTWARE REQUIREMENT

- Python (2.7)
- Packages Used:
    - Tensor flow
    - Keras
    - Numpy
    - Pickle
    - Scipy
    - Os

3.4. LANGUAGE SPECIFICATION

3.4.1. PYTHON

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python has a design philosophy that emphasizes code readability, notably using significant whitespace. It provides constructs that enable clear programming on both small and large scales. Van Rossum led the language community until stepping down as leader in July 2018.

Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural. It also has a comprehensive standard library.

Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of Python's other implementations. Python and CPython are managed by the non-profit Python Software Foundation.

Python is a multi-paradigm programming language. Object-oriented programming and structured programming are fully supported, and many of its features support functional programming and aspect-oriented programming (including by metaprogramming and metaobjects (magic methods)). Many other paradigms are supported via extensions, including design by contract and logic programming.

Python uses dynamic typing, and a combination of reference counting and a cycle-detecting garbage collector for memory management. It also features dynamic name resolution (late binding), which binds method and variable names during program execution. Python's design offers some support for functional programming in the Lisp tradition. It has filter(), map(), and reduce() functions; list comprehensions, dictionaries, sets and generator expressions. The standard library has two modules (itertools and functools) that implement functional tools borrowed from Haskell and Standard ML.

3.4.2. PACKAGES USED

TENSOR FLOW

TensorFlow is an end-to-end open source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications. It is a symbolic math library and is also used for machine learning applications such as neural networks.

KERAS

Keras is an open-source neural-network library written in Python. It is capable of running on top of TensorFlow, Microsoft Cognitive Toolkit, Theano, or PlaidML. Designed to enable fast experimentation with deep neural networks, it focuses on being user-friendly, modular and extensible. Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. It was developed with a focus on enabling fast experimentation

NUMPY

NumPy is a library for the Python programming language, adding support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays. NumPy is the fundamental package for scientific computing with Python. It contains among other things:

- a powerful N-dimensional array object
- sophisticated (broadcasting) functions
- tools for integrating C/C++ and Fortran code
- useful linear algebra, Fourier transform and random number capabilities

SCIPY

SciPy is a free and open-source Python library used for scientific computing and technical computing.SciPy contains modules for optimization, linear algebra, integration, interpolation, special functions, FFT, signal and image processing, ODE solvers and other tasks common in science and engineering.SciPy builds on the NumPy array object and is part of the NumPy stack which includes tools like Matplotlib, pandas and SymPy and an expanding set of scientific computing libraries.

3.5. METHODOLOGY

This system is a combination of GenNet and CultureNet which trained with multi culture data. CultureNet consists of seven subject-independent and subject-dependent GenNet and CultureNet models, trained with within-culture, cross-culture, mixed-culture and joint-culture data, where C0 indicates data from Japan and C1 indicates data from Serbia.

For all subject-independent models, a leave-one-child-out evaluation is used, where training is performed on 80% of each training child's data, validation on the remaining 20% of each training child's data and testing on 80% of the target child's data.

For all subject-dependent models, training is performed on 20% of each training child's data, validation on 20% of each validation child's data and testing on 80% of the target child's data. All experiments were repeated 10 times, each time starting from a different random initialization of the deep models.

Model 1 - Subject Independent, Within-Culture GenNet

The model is trained and tested on data of children from the same culture.

Model 2 - Subject Independent, Cross-Culture GenNet

The model is trained on data of children from C0 and tested on data of children from C1 and vice versa.

Model 3 - Subject Independent, Mixed-Culture GenNet

The model is trained on data from both cultures, then tested on each culture. Note: This is the first step of model 4.

Model 4 - Subject Independent, Joint-Culture CultureNet

The joint deep model is trained on data from both cultures, then the last layer is fine-tuned to each culture separately. Note: These are the first two steps of model 7

Model 5 - Subject Dependent, Within-Culture GenNet

The model is trained and tested on data of children from the same culture. Training data also includes 20% of target child data. Note: This is the subject-dependent version of model 1.

Model 6 - Subject Dependent, Child-Specific GenNet

The model is trained on 20% of target child data.

Model 7 - Subject Dependent, Joint-Culture CultureNet

The joint deep model is trained on data from both cultures, then the last layer is fine-tuned to each culture separately, then the last layer is additionally fine-tuned to each target child. The baseline generalized deep models (GenNet) are deep convolutional networks composed of layers of the ResNet, a pre-trained deep network, as well as additional network layers.

The culturalized deep models (CultureNet) follow the same architecture of GenNet; however, after training all layers with joint-culture data, the model freezes network parameters and uses culture-specific and/or child-specific data to fine-tune the last layer of the network. Face image data, which act as input for these models, are unidentifiable facial features, obtained in the output of the fine-tuned residual network (ResNet), a deep network optimized for object classification.

CHAPTER 4

SYSTEM DESIGN

4.1. SYSTEM ARCHITECTURE

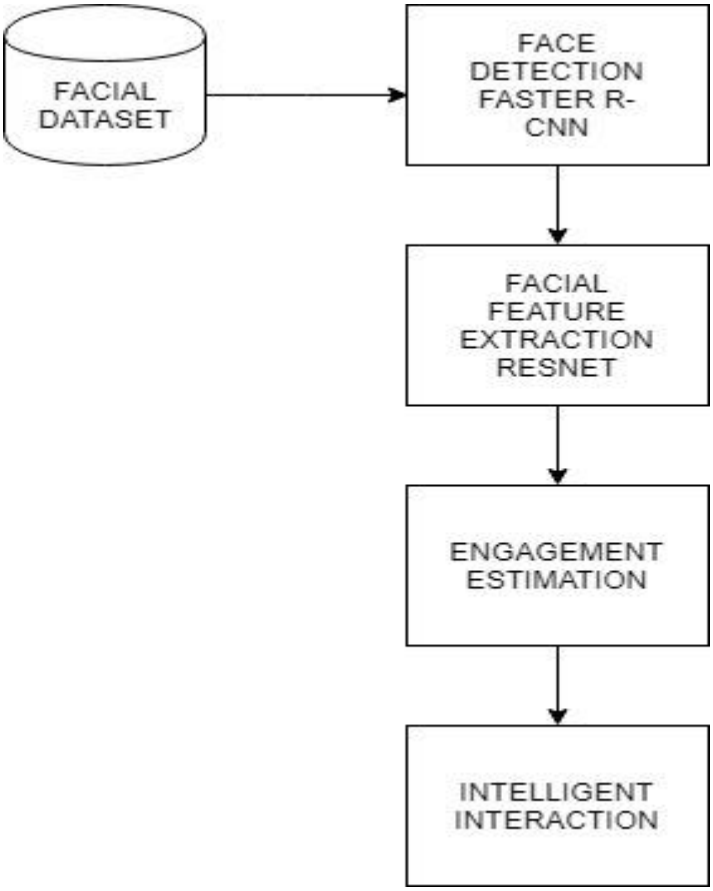


FIGURE 4.1. SYSTEM ARCHITECTURE

In figure 4.1, the system architectural design is the design process for identifying the subsystems making up the system and framework for subsystem control and communication. The goal of the architectural design is to establish the overall structure of software system.

4.2. USE CASE DIAGRAM

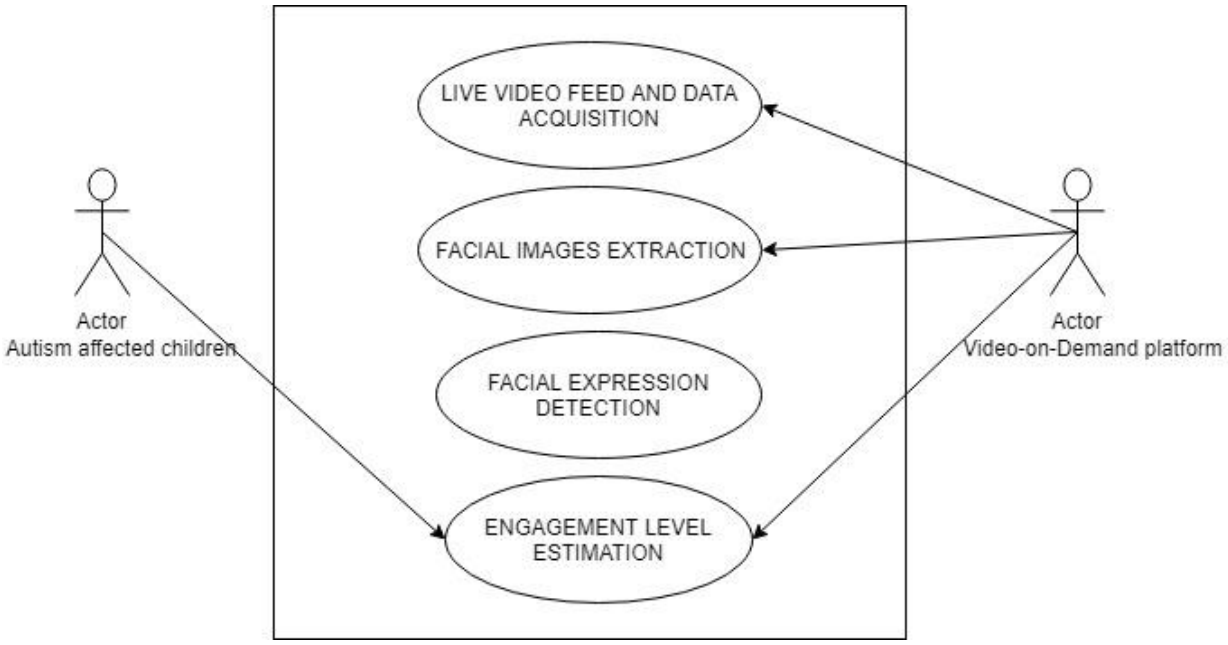


FIGURE 4.2: USE CASE DIAGRAM

In figure 4.2, Use case diagram represent the overall scenario of the system. A scenario is nothing but a sequence of steps describing an interaction between a user and a system. Thus use case is a set of scenario tied together by some goal. The use case diagram is drawn for exposing the functionalities of the system.

4.3. ACTIVITY DIAGRAM

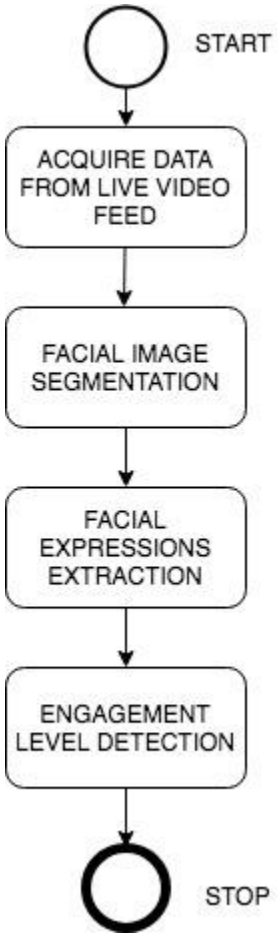


FIGURE 4.3: ACTIVITY DIAGRAM

In figure 4.3, the activity diagram shows a graphical representation for representing the flow of interaction within specific scenarios. It is similar to a flowchart in which various activities that can be performed in the system are represented.

4.4. DEPLOYMENT DIAGRAM

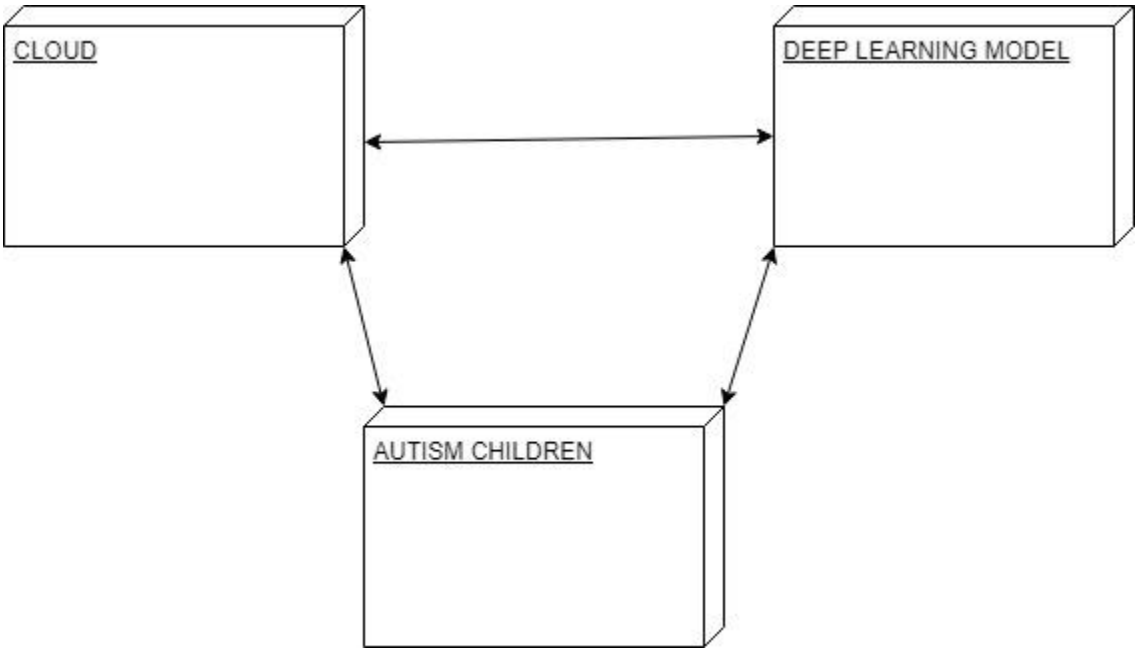


FIGURE 4.4: DEPLOYMENT DIAGRAM

In figure 4.4, a deployment diagram is a UML diagram type that shows the execution architecture of a system, including nodes such as hardware or software execution environments and the middleware connecting them. Deployment diagrams are typically used to visualize the physical hardware and software of a system. Using it you can understand how the system will be physically deployed on the hardware.

CHAPTER 5

MODULE DESCRIPTION

5.1. MODULES

- Data collection and data pickling
- Training the data
- Saving the weights and testing the model.

5.1.1. DATA COLLECTION AND PICKLING

The dataset has within-culture, cross-culture, mixed-culture and joint-culture data, where C0 indicates data from Japan and C1 indicates data from Serbia. The data is annotated with the engagement levels. In order to generalize the model multi culture data is obtained and annotated. The data is primarily facial data which includes facial expressions and the model should be trained with this dataset to data of the target culture and children when building deep models for engagement estimation. As it is a facial image data where the data is large and unlabelled. This data should be labelled and annotated initially before saving the dataset. Pickle is used to save the dataset and 80% of the dataset is held as training dataset and 20% of the dataset is held as validation set and the model can load the dataset by using the pickled dataset.

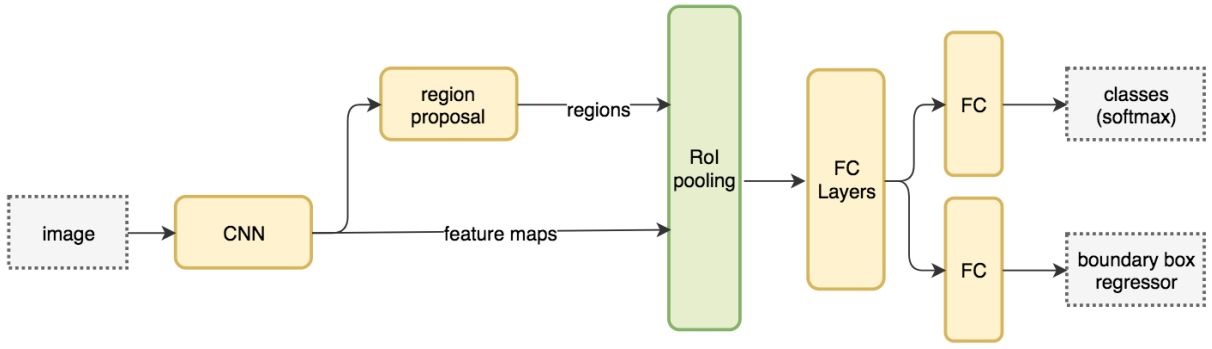
5.1.2. TRAINING THE DATASET

The dataset with facial expressions are loaded and it is unloaded from the pickled data. For all subject-independent models, a leave-one-child-out evaluation, where we performed training on 80% of each training child's data, validation on

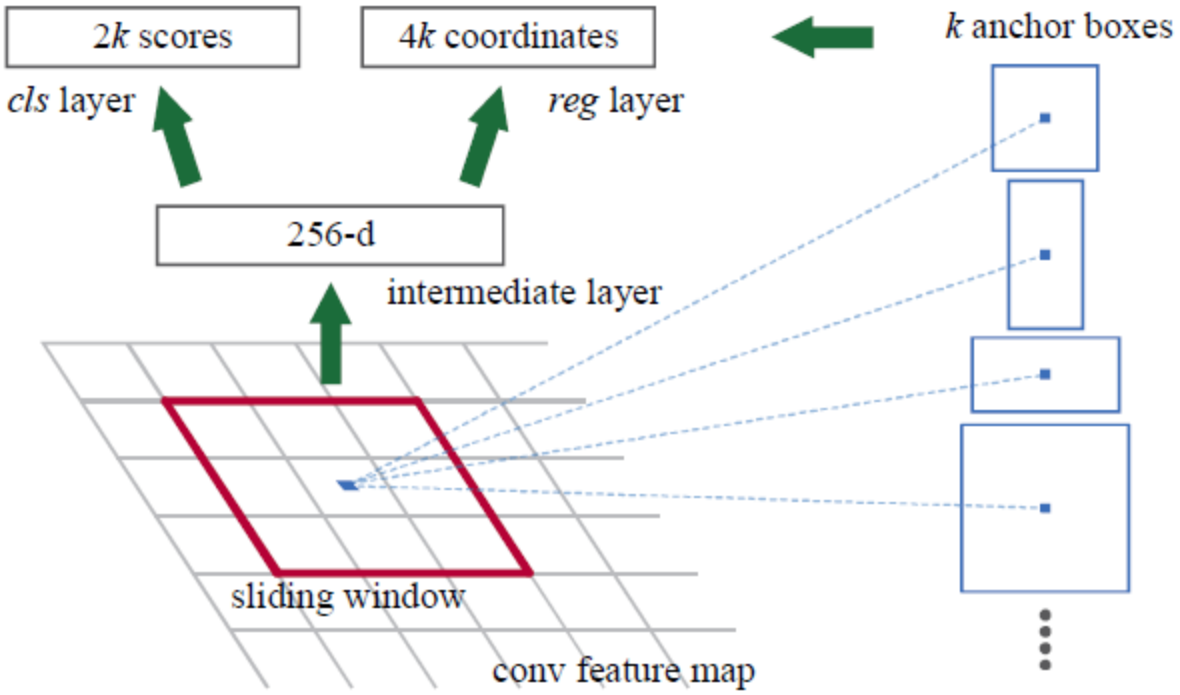
the remaining 20% of each training child's data and testing on 80% of the target child's data is used. For all subject-dependent models, we performed training on 20% of each training child's data, validation on 20% of each validation child's data and testing on 80% of the target child's data. All experiments were repeated 10 times, each time starting from a different random initialization of the deep models. The dataset is trained with an architecture of faster R-CNN and resnet together stacked with.

FASTER R-CNN

Faster R-CNN has two networks: region proposal network (RPN) for generating region proposals and a network using these proposals to detect objects**.**In Faster R-CNN , both region proposal generation and objection detection tasks are all done by the same conv networks. With such design, object detection is much faster. In Faster R-CNN , RPN using SS is replaced by RPN using CNN. And this CNN is shared with detection network. This CNN can be ZFNet or VGGNet. Thus, the overall network is as below:



1. First, the picture goes through conv layers and feature maps are extracted.
2. Then a sliding window is used in RPN for each location over the feature map.
3. For each location, k (k=9) anchor boxes are used (3 scales of 128, 256 and 512, and 3 aspect ratios of 1:1, 1:2, 2:1) for generating region proposals.
4. A *cls* layer outputs *2k* scores whether there is object or not for *k* boxes.
5. A *reg* layer outputs *4k* for the coordinates (box center coordinates, width and height) of *k* boxes.
6. With a size of *W*×*H* feature map, there are *WHk* anchors in total.



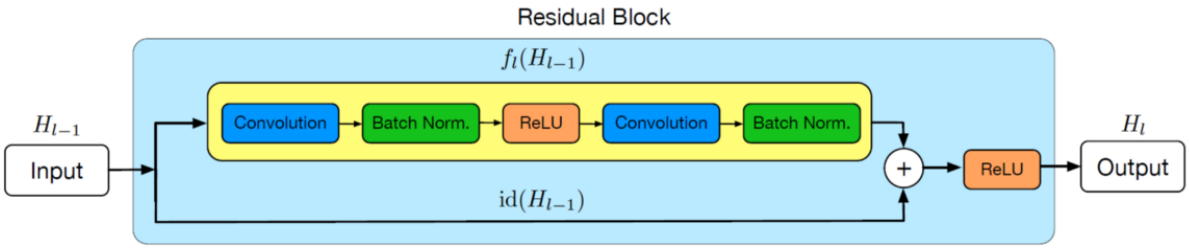
Faster R-CNN fixes the problem of selective search by replacing it with Region Proposal Network (RPN). We first extract feature maps from the input image using ConvNet and then pass those maps through a RPN which returns object proposals. Finally, these maps are classified and the bounding boxes are predicted.

1. Take an input image and pass it to the ConvNet which returns feature maps for the image
2. Apply Region Proposal Network (RPN) on these feature maps and get object proposals
3. Apply ROI pooling layer to bring down all the proposals to the same size
4. Finally, pass these proposals to a fully connected layer in order to classify any predict the bounding boxes for the image

RESNET

A residual neural network (ResNet) is an artificial neural network (ANN) of a kind that builds on constructs known from pyramidal cells in the cerebral cortex. Residual neural networks do this by utilizing skip connections, or shortcuts to jump over some layers. Typical ResNet models are implemented with single-layer skips. neural network without residual parts explores more of the feature space. This makes it more vulnerable to perturbations that cause it to leave the manifold, and necessitates extra training data to recover. The flow graph of resnet architecture is given as,An additional weight matrix may be used to learn the skip weights; these models are known as HighwayNets. Models with several parallel skips are referred to as DenseNets. In the context of residual neural networks, a non-residual network may be described as a plain network. One motivation for skipping over layers is to avoid the problem of vanishing gradients, by reusing activations from a previous layer until the adjacent layer learns its weights. During training, the weights adapt

to mute the upstream layer, and amplify the previously-skipped layer. In the simplest case, only the weights for the adjacent layer's connection are adapted, with no explicit weights for the upstream layer. This works best when a single non-linear layer is stepped over, or when the intermediate layers are all linear. If not, then an explicit weight matrix should be learned for the skipped connection. Skipping effectively simplifies the network, using fewer layers in the initial training stages. This speeds learning by reducing the impact of vanishing gradients, as there are fewer layers to propagate through. The network then gradually restores the skipped layers as it learns the feature space. Towards the end of training, when all layers are expanded, it stays closer to the manifold and thus learns faster. The block diagram of resnet are as follows



PSEUDOCODE FOR RESNET:

```
TRAIN_TEST(train_features, test_features, unlabeled_features)
Returns test results of the deep classifier.
    ResNet ← load()                                  ▷ ResNet50 weights
    Incep ← load()                                   ▷ InceptionV3 weights
    Xcep ← load()                                    ▷ Xception weights
    res_features ← ResNet(train_features,
    test_features, unlabeled_features)
    inc_features ← Incep(train_features,
    test_features, unlabeled_features)
    xcp_features ← Xcep(train_features,
    test_features, unlabeled_features)
    all_features ← concat(res_features, inc_features, xcp_features
    if save_features = True then
        savef(all_features)
    end if
    model ← create_model()
    if training = True then
        fitted_model ← model.fit(all_features[train], labels)
        if pseudo = True then
            newly_labeled ←
            pseudo_labeling(fitted_model, all_features[unlabeled])
            fitted_model ← fitted_model.fit(newly_labeled, labels)
        end if
        model ← fitted_model
    end if
    if testing = True then
        results ← fitted_model(all_features[test])
    end if
    return results
end
```

5.1.3. SAVING THE WEIGHTS AND TESTING THE MODEL

The weights are gained from training of the joint culture data and the weights are stored in .h5 format. After training the dataset completely, the weights are stored simultaneously and the training is done with the multi culture data where

the engagement level is noted. The pretrained weights can be noted and used with new testing data. The model shows good accuracy for both culture data (Asia and Europe)

These pre-trained networks demonstrate a strong ability to generalize to images outside the ImageNet dataset via transfer learning. We make modifications in the pre-existing model by fine-tuning the model. Since we assume that the pre-trained network has been trained quite well, we would not want to modify the weights too soon and too much. While modifying we generally use a learning rate smaller than the one used for initially training the model.

To properly evaluate a model, you hold out a sample of data that has been labeled with the target (ground truth) from the training datasource. Evaluating the predictive accuracy of an ML model with the same data that was used for training is not useful, because it rewards models that can "remember" the training data, as opposed to generalizing from it. Once you have finished training the ML model, you send the model the held-out observations for which you know the target values. You then compare the predictions returned by the ML model against the known target value.

CHAPTER 6

CONCLUSION AND FUTURE ENHANCEMENT

6.1. CONCLUSION

In this work, different deep learning settings for automated estimation of engagement from face images of children with ASC, who participated in one-day robot-assisted autism therapy are investigated. These results reveal important findings in terms of how deep learning can be used to leverage face-image data of children with ASC to attain better estimation of target engagement. More specifically, the role of the cultural label as the driving factor in learning deep models for their generalization to different children with ASC: within and across two cultures (Asia vs. Europe) is identified. Furthermore, analysis in the child-independent and child-dependent manner is verified. Due to the large difference in the distribution of engagement levels in the two cultures, the deep models trained on only one culture have limited ability to generalize to the other culture. . This indicates the importance of having access to data of the target culture and children when building deep models for engagement estimation.

6.2. FUTURE ENHANCEMENT

This system primarily involves estimating the engagement level of the autism affected child and the main parameter we set is facial expressions. Parameters like body temperature, Speech can be recorded by sensors and the complete characteristic traits like sensing, perception and interaction can be estimated for the betterment of Autism affected children

APPENDIX 1

SAMPLE CODING

genNet.py

```
from utils import icc, ccc, mae

from scipy.stats import pearsonr as pcc
from keras.models import Model
from keras.layers import Dense, Input
from keras.callbacks import EarlyStopping
import os
import tensorflow as tf
import numpy as np
os.environ['PYTHONHASHSEED'] = '0'
os.environ['CUDA_VISIBLE_DEVICES'] = '0, 1'
np.random.seed(42)
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,
inter_op_parallelism_threads=1)
import random as rn
rn.seed(12345)
tf.set_random_seed(1234)
class genNet(Model):
    def __init__(self, input_dim, trainable=[True,True,True,True,True], lsize0=500,
lsize1=200, lsize2=100, lsize3=50, epochs=20, batch_size=128, **kwargs):
        self.epochs = epochs
        self.batch_size = batch_size
        inp = Input(shape=(input_dim,))
```

```
        l1 = Dense(lsize0, activation='relu', trainable=trainable[0])(inp)
        l2 = Dense(lsize1, activation='relu', trainable=trainable[1])(l1)
        l3 = Dense(lsize2, activation='relu', trainable=trainable[2])(l2)
        l4 = Dense(lsize3, activation='relu', trainable=trainable[3])(l3)
        l5 = Dense(1, activation='linear', trainable=trainable[4])(l4)
        super(genNet, self).__init__(inputs=[inp], outputs=[l5])
    def train_model(self, x_train, y_train, x_val, y_val, dtype):
        self.compile(optimizer='adadelta', loss='mean_squared_error',
metrics=['mae'])
        stopper = EarlyStopping(monitor='val_loss', min_delta=0.01, patience=4,
mode='auto')
        self.fit(x_train, y_train, epochs=self.epochs, batch_size=self.batch_size,
verbose=1, callbacks=[stopper], validation_data=(x_val,y_val))
        self.save_weights('Weights/{}_weights.h5'.format(dtype))
    def make_report(self, report_name, id_test, x_test, y_test, country_test,
frame_test):
        if not os.path.exists('Reports/' + report_name):
            os.mkdir('Reports/' + report_name)
        results = self.predict(x_test)
        header = 'Country,Child,Frame'
        for output_layer in self.get_config()['output_layers']:
            header += ',{}_Actual'.format(output_layer[0])
        for output_layer in self.get_config()['output_layers']:
            header += ',{}_Prediction'.format(output_layer[0])
        header += '\n'
        with open('Reports/{}/evaluation_report.txt'.format(report_name), 'a') as f:
```

```
        if os.stat('Reports/{}/evaluation_report.txt'.format(report_name)).st_size ==
0:
            f.write(header)
        for row in range(len(results)):
            entry = ','.join([str(i) for i in country_test[row]]) + ','
            entry += ','.join([str(i) for i in id_test[row]]) + ','
            entry += ','.join([str(i) for i in frame_test[row]]) + ','
            entry += ','.join([str(i) for i in y_test[row]]) + ','
            entry += ','.join([str(i) for i in results[row]]) + '\n'
            f.write(entry)
    cultures = np.unique(country_test)
    for c in cultures:
        culture_rows = np.where(country_test == c)[0]
        culture_ids = id_test[culture_rows] # get ID rows for culture c
        unique_ids = np.unique(culture_ids) # get unique IDs for culture c
        for u in unique_ids:
            all_id_rows = np.where(id_test == u)[0]
            id_rows = np.intersect1d(all_id_rows, culture_rows)
            id_icc = icc(results[id_rows], y_test[id_rows])[0]
            id_pcc = pcc(results[id_rows], y_test[id_rows])[0][0]
            id_ccc = ccc(results[id_rows], y_test[id_rows])
            id_mae = mae(results[id_rows], y_test[id_rows])
            icc_entry = '{},{},{}\n'.format(c, u, id_icc)
            pcc_entry = '{},{},{}\n'.format(c, u, id_pcc)
            ccc_entry = '{},{},{}\n'.format(c, u, id_ccc)
            mae_entry = '{},{},{}\n'.format(c, u, id_mae)
            with open('Reports/{}/icc_report.txt'.format(report_name), 'a') as f:
```

```
            f.write(icc_entry)
        with open('Reports/{}/pcc_report.txt'.format(report_name), 'a') as f:
            f.write(pcc_entry)
        with open('Reports/{}/ccc_report.txt'.format(report_name), 'a') as f:
            f.write(ccc_entry)
        with open('Reports/{}/mae_report.txt'.format(report_name), 'a') as f:
            f.write(mae_entry)
    return results
if __name__ == '__main__':
    pass
```

models.py

```
from genNet import genNet
from keras import backend as K
import os
import numpy as np
import tensorflow as tf
os.environ['PYTHONHASHSEED'] = '0'
os.environ['CUDA_VISIBLE_DEVICES'] = '0, 1'
np.random.seed(42)
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,
inter_op_parallelism_threads=1)
import random as rn
rn.seed(12345)
tf.set_random_seed(1234)
def __run_gen_net(data, dtype, trainable=[True,True,True,True,True],
weights=None, **kwargs):
```

```
        sess = tf.Session(graph=tf.get_default_graph(), config=session_conf)
    K.set_session(sess)
    _, _, id_test, x_train, x_val, x_test, y_train, y_val, y_test, _, _, culture_test, _, _,
frame_test = data
 # Build model
    model = genNet(input_dim=x_train.shape[1], trainable=trainable)
    if weights is None:
        model.train_model(x_train, y_train, x_val, y_val, dtype)
    else:
        model.set_weights(weights)
        model.train_model(x_train, y_train, x_val, y_val, dtype)
    _ = model.make_report('ExperimentdS_deep_'+dtype, id_test, x_test, y_test,
culture_test, frame_test)
optimized_weights = model.get_weights()
    K.clear_session()
    return optimized_weights
def __run_culture_net(data_cA, data_cB, prelim_data, dtype_prelim_cA,
dtype_final_cA):
    # Build preliminary model
    prelim_weights = __run_gen_net(prelim_data, dtype_prelim_cA)
    # Build culture-specific model (for culture A)
    _ = __run_gen_net(data_cA, dtype_final_cA,
trainable=[False,False,False,False,True], weights=prelim_weights)
    return None
def run_m1(c0_data, c1_data):
    print('---------- Running Model 1 ----------')
    _ = __run_gen_net(c0_data, 'm1')
```

```
    _ = __run_gen_net(c1_data, 'm1')
    print('---------- Completed Model 1 ----------')
    return None
def run_m2(c0_data, c1_data):
    print('---------- Running Model 2 ----------')
 c0_m2_data = (c1_data[0], c0_data[1], c0_data[2], c1_data[3], c0_data[4],
c0_data[5], c1_data[6], c0_data[7], c0_data[8], c1_data[9], c0_data[10],
c0_data[11], c1_data[12], c0_data[13], c0_data[14])
    _ = __run_gen_net(c0_m2_data, 'm2')
    c1_m2_data = (c0_data[0], c1_data[1], c1_data[2], c0_data[3], c1_data[4],
c1_data[5], c0_data[6], c1_data[7], c1_data[8], c0_data[9], c1_data[10],
c1_data[11], c0_data[12], c1_data[13], c1_data[14])
    _ = __run_gen_net(c1_m2_data, 'm2')
    print('---------- Completed Model 2 ----------')
    return None
def run_m3(c0_data_merged, c1_data_merged):
    print('---------- Running Model 3 ----------')
    c0_m3_weights = __run_gen_net(c0_data_merged, 'm3')
    c1_m3_weights = __run_gen_net(c1_data_merged, 'm3')
    print('---------- Completed Model 3 ----------')
    return c0_m3_weights, c1_m3_weights
def run_m4(c0_data, c1_data, c0_data_merged, c1_data_merged, c0_m3_weights,
c1_m3_weights):
    print('---------- Running Model 4 ----------')
    if c0_m3_weights is None:
        __run_culture_net(c0_data, c1_data, c0_data_merged, 'm4_prelim', 'm4')
    else:
```

```
        _ = __run_gen_net(c0_data, 'm4', trainable=[False,False,False,False,True],
weights=c0_m3_weights)
    if c1_m3_weights is None:
        __run_culture_net(c1_data, c0_data, c1_data_merged, 'm4_prelim', 'm4')
    else:
        _ = __run_gen_net(c1_data, 'm4', trainable=[False,False,False,False,True],
weights=c1_m3_weights)
    print('---------- Completed Model 4 ----------')
    return None
def run_m5(c0_data_targetRep, c1_data_targetRep):
      print('---------- Running Model 5 ----------')
    _ = __run_gen_net(c0_data_targetRep, 'm5')
    _ = __run_gen_net(c1_data_targetRep, 'm5')
    print('---------- Completed Model 5 ----------')
    return None
def run_m6(c0_data_targetOnly, c1_data_targetOnly):
    print('---------- Running Model 6 ----------')
    _ = __run_gen_net(c0_data_targetOnly, 'm6')
    _ = __run_gen_net(c1_data_targetOnly, 'm6')
    print('---------- Completed Model 6 ----------')
    return None
def run_prelim_m7(m7_joint_data, c0_data_All, c1_data_All):
    print('---------- Running Preliminary Model 7 ----------')
    prelim_weights = __run_gen_net(m7_joint_data, 'm7_joint_prelim')
    c0_m7_prelim_weights = __run_gen_net(c0_data_All, 'm7_prelim',
trainable=[False,False,False,False,True], weights=prelim_weights)
```

```
    c1_m7_prelim_weights = __run_gen_net(c1_data_All, 'm7_prelim',
trainable=[False,False,False,False,True], weights=prelim_weights)
    print('---------- Completed Preliminary Model 7 ----------')
    return c0_m7_prelim_weights, c1_m7_prelim_weights
def run_m7(c0_data_targetOnly, c1_data_targetOnly, c0_m7_prelim_weights,
c1_m7_prelim_weights):
    print('---------- Running Model 7 ----------')
    _ = __run_gen_net(c0_data_targetOnly, 'm7',
trainable=[False,False,False,False,True], weights=c0_m7_prelim_weights)
    _ = __run_gen_net(c1_data_targetOnly, 'm7',
trainable=[False,False,False,False,True], weights=c1_m7_prelim_weights)
    print('---------- Completed Model 7 ----------')
    return None
if __name__ == '__main__':
    pass
```

runIROS.py

```
from utils import load_data, leave_1_out_ids, all_children_ids, target_only_ids,
merge_data, process_summary
from models import *
import os
import pathlib
import tensorflow as tf
import numpy as np
os.environ['PYTHONHASHSEED'] = '0'
os.environ['CUDA_VISIBLE_DEVICES'] = '0, 1'
np.random.seed(42)
```

```
session_conf = tf.ConfigProto(intra_op_parallelism_threads=1,
inter_op_parallelism_threads=1)
import random as rn
rn.seed(12345)
tf.set_random_seed(1234)
if __name__ == '__main__':
    CURRENT_DIR = os.path.dirname(os.path.abspath(__file__))
  REPORTS_FOLDER_DIR = os.path.join(CURRENT_DIR, 'Reports')
  WEIGHTS_FOLDER_DIR = os.path.join(CURRENT_DIR, 'Weights')
  pathlib.Path(REPORTS_FOLDER_DIR).mkdir(parents=True, exist_ok=True)
  pathlib.Path(WEIGHTS_FOLDER_DIR).mkdir(parents=True, exist_ok=True)
  c0_IDs = [1,2,3,4,6,7,8,9,10,11,12,13,14,16,17] # Culture index 0
  c1_IDs = [2,3,4,5,6,7,8,9,10,13,14,15,17,18,20] # Culture index 1
  c0_IDs_1Out = leave_1_out_ids(c0_IDs)
  c1_IDs_1Out = leave_1_out_ids(c1_IDs)
  c0_IDs_All = [c0_IDs]*3
  c1_IDs_All = [c1_IDs]*3
  c0_IDs_targetRep = all_children_ids(c0_IDs)
  c1_IDs_targetRep = all_children_ids(c1_IDs)
  c0_IDs_targetOnly = target_only_ids(c0_IDs)
  c1_IDs_targetOnly = target_only_ids(c1_IDs)
  c0_data_All = load_data(c0_IDs_All, 0, data_proportion=[0.2,0,0.2,0.8])
  c1_data_All = load_data(c1_IDs_All, 1, data_proportion=[0.2,0,0.2,0.8])
  m7_joint_data = []
  for p in range(len(c0_data_All)):
    m7_joint_data.append(np.concatenate((c0_data_All[p], c1_data_All[p]),
axis=0))
```

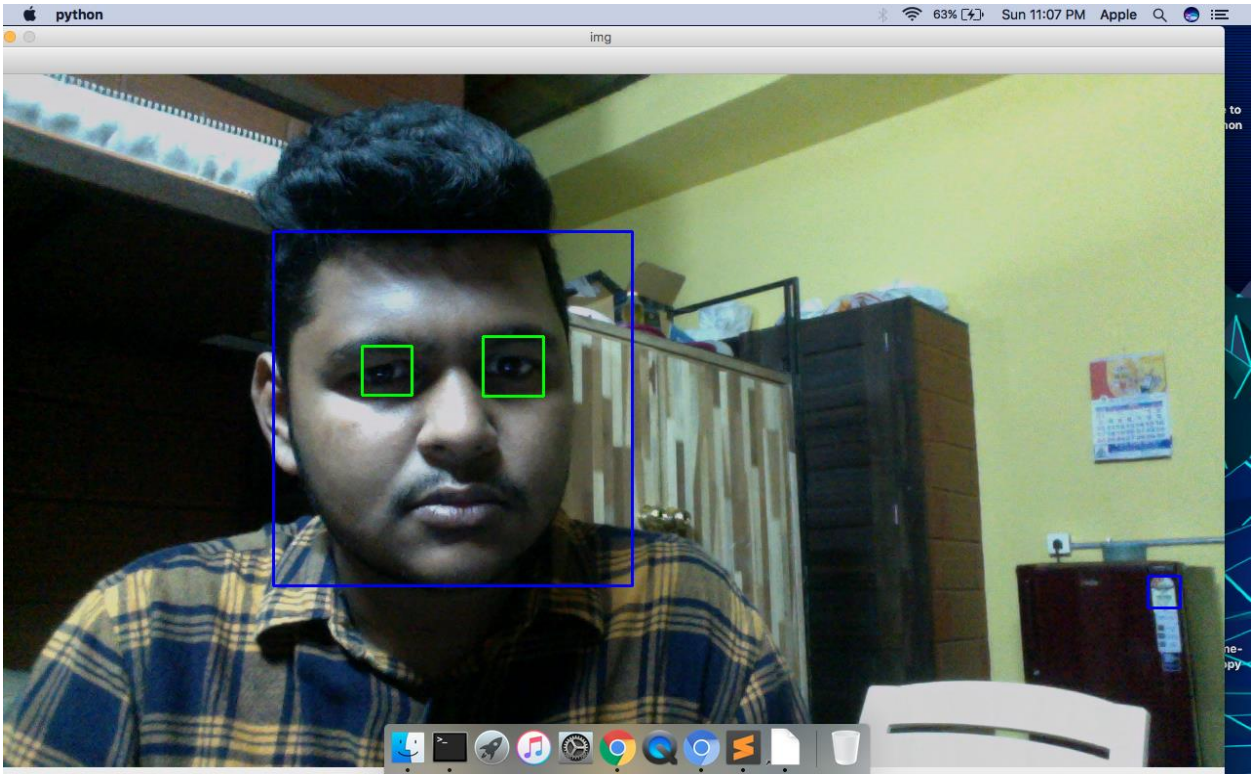
```
    m7_joint_data = tuple(m7_joint_data)
    Preliminary Model 7 - Joint Culture / SD:
    c0_m7_prelim_weights, c1_m7_prelim_weights =
run_prelim_m7(m7_joint_data, c0_data_All, c1_data_All)
    for i in range(len(c0_IDs_1Out)):
        c0_data = load_data(c0_IDs_1Out[i], 0, data_proportion=[0.8,0.8,1,0.8])
        c1_data = load_data(c1_IDs_1Out[i], 1, data_proportion=[0.8,0.8,1,0.8])
        c0_data_targetRep = load_data(c0_IDs_targetRep[i], 0,
data_proportion=[0.2,0,0.2,0.8])
        c1_data_targetRep = load_data(c1_IDs_targetRep[i], 1,
data_proportion=[0.2,0,0.2,0.8])
        c0_data_targetOnly = load_data(c0_IDs_targetOnly[i], 0,
data_proportion=[0.2,0,0.2,0.8])
        c1_data_targetOnly = load_data(c1_IDs_targetOnly[i], 1,
data_proportion=[0.2,0,0.2,0.8])
        c0_data_merged = merge_data(c0_data, c1_data)
        c1_data_merged = merge_data(c1_data, c0_data)
        for loop in range(10):
            print('---------- CHILD {} ----------'.format(i+1))
            print('---------- FOLD {} ----------'.format(loop+1))
            c0_m3_weights = None
            c1_m3_weights = None
            run_m1(c0_data, c1_data)
            run_m2(c0_data, c1_data)
            c0_m3_weights, c1_m3_weights = run_m3(c0_data_merged,
c1_data_merged)
```

```
        run_m4(c0_data, c1_data, c0_data_merged, c1_data_merged,
c0_m3_weights, c1_m3_weights)
        run_m5(c0_data_targetRep, c1_data_targetRep)
        run_m6(c0_data_targetOnly, c1_data_targetOnly)
        run_m7(c0_data_targetOnly, c1_data_targetOnly, c0_m7_prelim_weights,
c1_m7_prelim_weights)
    process_summary(REPORTS_FOLDER_DIR)
```

APPENDIX 2

SCREENSHOTS

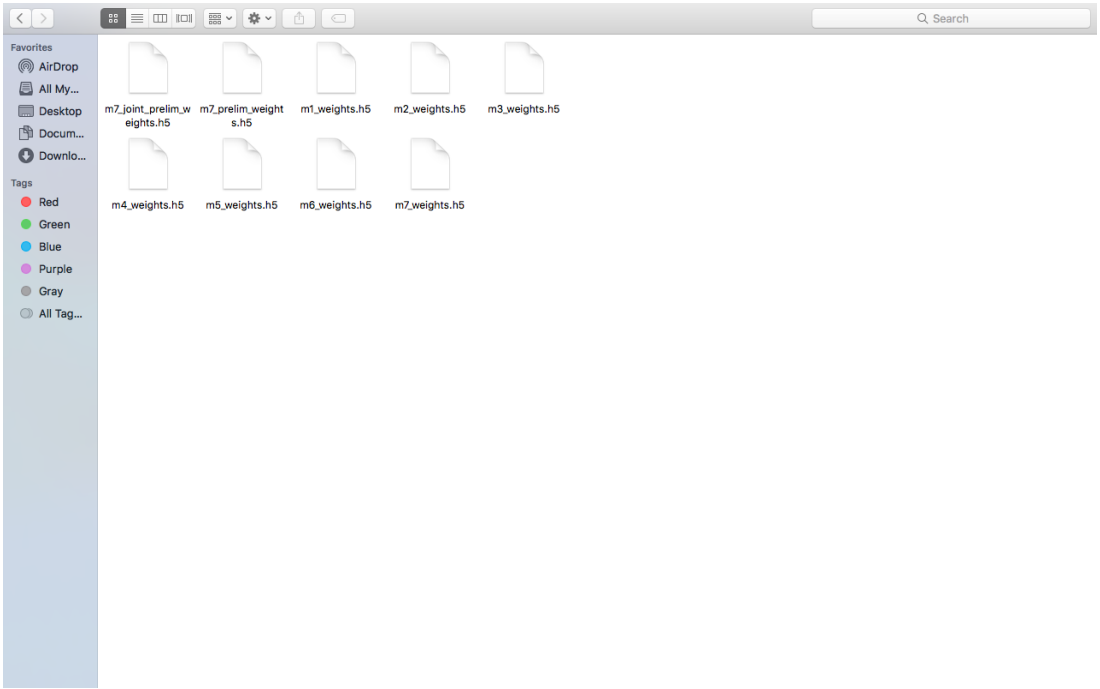
Detection of images from live video feed



Training the dataset

```
Epoch 2/20
67200/67200 [==============================] - 25s 376us/step - loss: 0.0728 - mean_absolute_error: 0.1700 - val_loss: 0.4606 - val_mean_absolute_error: 0.5531
Epoch 3/20
67200/67200 [==============================] - 25s 373us/step - loss: 0.0604 - mean_absolute_error: 0.1502 - val_loss: 0.4228 - val_mean_absolute_error: 0.5270
Epoch 4/20
67200/67200 [==============================] - 25s 374us/step - loss: 0.0508 - mean_absolute_error: 0.1359 - val_loss: 0.4479 - val_mean_absolute_error: 0.5426
Epoch 5/20
67200/67200 [==============================] - 25s 374us/step - loss: 0.0437 - mean_absolute_error: 0.1261 - val_loss: 0.3745 - val_mean_absolute_error: 0.4910
---------- Completed Model 2 ----------
---------- Running Model 3 ----------
Train on 156001 samples, validate on 16800 samples
Epoch 1/20
156001/156001 [==============================] - 3696s 24ms/step - loss: 0.1165 - mean_absolute_error: 0.2566 - val_loss: 0.0978 - val_mean_absolute_error: 0.2335
Epoch 2/20
156001/156001 [==============================] - 64s 410us/step - loss: 0.0795 - mean_absolute_error: 0.1995 - val_loss: 0.0657 - val_mean_absolute_error: 0.1611
Epoch 3/20
156001/156001 [==============================] - 64s 409us/step - loss: 0.0677 - mean_absolute_error: 0.1811 - val_loss: 0.0605 - val_mean_absolute_error: 0.1596
Epoch 4/20
156001/156001 [==============================] - 61s 394us/step - loss: 0.0599 - mean_absolute_error: 0.1693 - val_loss: 0.0545 - val_mean_absolute_error: 0.1455
Epoch 5/20
156001/156001 [==============================] - 61s 388us/step - loss: 0.0538 - mean_absolute_error: 0.1598 - val_loss: 0.0549 - val_mean_absolute_error: 0.1366
Epoch 6/20
156001/156001 [==============================] - 63s 402us/step - loss: 0.0490 - mean_absolute_error: 0.1520 - val_loss: 0.0470 - val_mean_absolute_error: 0.1189
Epoch 7/20
156001/156001 [==============================] - 2846s 18ms/step - loss: 0.0447 - mean_absolute_error: 0.1450 - val_loss: 0.0483 - val_mean_absolute_error: 0.1218
Epoch 8/20
156001/156001 [==============================] - 1515s 10ms/step - loss: 0.0412 - mean_absolute_error: 0.1389 - val_loss: 0.0434 - val_mean_absolute_error: 0.1142
Epoch 9/20
156001/156001 [==============================] - 64s 407us/step - loss: 0.0381 - mean_absolute_error: 0.1334 - val_loss: 0.0404 - val_mean_absolute_error: 0.1115
Epoch 10/20
156001/156001 [==============================] - 62s 398us/step - loss: 0.0351 - mean_absolute_error: 0.1282 - val_loss: 0.0401 - val_mean_absolute_error: 0.1195
Epoch 11/20
156001/156001 [==============================] - 63s 404us/step - loss: 0.0328 - mean_absolute_error: 0.1239 - val_loss: 0.0420 - val_mean_absolute_error: 0.1190
Epoch 12/20
156001/156001 [==============================] - 64s 411us/step - loss: 0.0304 - mean_absolute_error: 0.1195 - val_loss: 0.0379 - val_mean_absolute_error: 0.1062
Train on 156001 samples, validate on 16800 samples
Epoch 1/20
156001/156001 [==============================] - 24698s 158ms/step - loss: 0.1123 - mean_absolute_error: 0.2448 - val_loss: 0.0973 - val_mean_absolute_error: 0.2468
Epoch 2/20
156001/156001 [==============================] - 65s 415us/step - loss: 0.0776 - mean_absolute_error: 0.1919 - val_loss: 0.0832 - val_mean_absolute_error: 0.2235
Epoch 3/20
156001/156001 [==============================] - 66s 426us/step - loss: 0.0656 - mean_absolute_error: 0.1742 - val_loss: 0.0770 - val_mean_absolute_error: 0.2136
Epoch 4/20
156001/156001 [==============================] - 65s 415us/step - loss: 0.0576 - mean_absolute_error: 0.1623 - val_loss: 0.0821 - val_mean_absolute_error: 0.2165
Epoch 5/20
156001/156001 [==============================] - 63s 402us/step - loss: 0.0519 - mean_absolute_error: 0.1532 - val_loss: 0.0752 - val_mean_absolute_error: 0.2050
Epoch 6/20
156001/156001 [==============================] - 63s 406us/step - loss: 0.0467 - mean_absolute_error: 0.1447 - val_loss: 0.0698 - val_mean_absolute_error: 0.1985
Epoch 7/20
156001/156001 [==============================] - 61s 393us/step - loss: 0.0426 - mean_absolute_error: 0.1379 - val_loss: 0.0718 - val_mean_absolute_error: 0.1945
Epoch 8/20
156001/156001 [==============================] - 61s 391us/step - loss: 0.0390 - mean_absolute_error: 0.1320 - val_loss: 0.0653 - val_mean_absolute_error: 0.1867
Epoch 9/20
156001/156001 [==============================] - 3662s 23ms/step - loss: 0.0359 - mean_absolute_error: 0.1268 - val_loss: 0.0655 - val_mean_absolute_error: 0.1852
Epoch 10/20
 81792/156001 [==============>...............] - ETA: 30s - loss: 0.0331 - mean_absolute_error: 0.1220
```

Gaining and saving the weights



Estimation of Engagement Levels

```
Country,Child,Frame,dense_5_Actual,dense_5_Prediction
0.0,1.0,2704.0,0.78,-0.28946105
0.0,1.0,4757.0,-0.329,0.96016264
0.0,1.0,5667.0,0.0,-0.27579334
0.0,1.0,10457.0,0.622,0.9177253
0.0,1.0,9048.0,0.803,0.80486834
0.0,1.0,7294.0,0.559,0.89918804
0.0,1.0,9782.0,0.843,0.8781115
0.0,1.0,16326.0,0.827,0.9270012
0.0,1.0,9298.0,-0.289,0.9292232
0.0,1.0,2884.0,0.78,0.77991545
0.0,1.0,7910.0,0.628,0.85731936
0.0,1.0,4273.0,0.78,0.9052229
0.0,1.0,10465.0,0.634,0.9508349
0.0,1.0,10064.0,0.661,0.82889235
0.0,1.0,7219.0,0.503,0.8577348
0.0,1.0,18264.0,0.401,0.92176044
0.0,1.0,8434.0,0.882,0.85481465
0.0,1.0,15428.0,0.259,0.65130126
0.0,1.0,17472.0,-0.566,0.27545106
0.0,1.0,16150.0,0.882,0.8089242
0.0,1.0,5025.0,0.843,0.927263
0.0,1.0,8122.0,0.78,0.71197355
0.0,1.0,15288.0,-0.329,0.8079605
0.0,1.0,12818.0,-0.044,-0.15102541
0.0,1.0,16250.0,0.866,0.95543766
0.0,1.0,18381.0,0.0,0.22432816
0.0,1.0,14376.0,0.0,0.6474153
0.0,1.0,10645.0,-0.345,0.8788909
0.0,1.0,10344.0,0.661,0.93228626
0.0,1.0,7570.0,0.888,0.82040906
0.0,1.0,7432.0,0.78,0.9124019
0.0,1.0,9375.0,-0.402,0.86800313
0.0,1.0,7618.0,0.843,0.73048913
0.0,1.0,13576.0,0.645,0.7913979
0.0,1.0,14073.0,0.401,0.9005095
0.0,1.0,4670.0,-0.083,0.8560935
0.0,1.0,13946.0,-0.432,0.32252645
0.0,1.0,3240.0,0.701,0.89581406
0.0,1.0,19063.0,0.78,0.67061985
0.0,1.0,13817.0,-0.329,0.96619666
0.0,1.0,6623.0,0.361,0.09354375
0.0,1.0,14574.0,0.882,0.9130682
0.0,1.0,12528.0,0.559,0.844864
0.0,1.0,7254.0,0.503,0.76613855
0.0,1.0,4921.0,0.882,0.7300929
0.0,1.0,9752.0,0.622,0.765646
0.0,1.0,10967.0,-0.068,0.94260275
0.0,1.0,4180.0,0.606,0.79171824
0.0,1.0,15233.0,0.906,0.9294932
0.0,1.0,16111.0,0.866,0.8817738
0.0,1.0,12657.0,0.685,0.7472371
0.0,1.0,9069.0,0.066,0.87538207
0.0,1.0,16241.0,0.866,0.955171
0.0,1.0,15572.0,-0.432,0.84239197
0.0,1.0,7257.0,0.503,0.6609534
0.0,1.0,8939.0,0.401,0.8726959
0.0,1.0,10079.0,0.622,0.9255371
0.0,1.0,7906.0,0.712,0.9127898
0.0,1.0,12802.0,-0.147,0.8134316
0.0,1.0,15429.0,0.282,0.8843423
0.0,1.0,10162.0,0.464,0.9259362
```

REFERENCES

[1] M. Abadi et al., “Tensorflow: A system for large-scale machine learning,” in USENIX Symposium on OSDI, 2016.

[2] L. B. Adamson et al., “Early interests and joint engagement in typical development, autism and down syndrome,” Journal of Autism and Developmental Disorders, 2010.

[3] D. Almirall, C. Kasari, D. F. McCaffrey and I. Nahum-Shani, “Developing optimized adaptive interventions in education,” Journal of Research on Educational Effectiveness, 2018.

[4] S. M. Anzalone et al., “Evaluating the engagement with social robots,” IJSR, 2015.

[5] A. P. Association, Diagnostic and Statistical Manual of Mental Disorders (DSM-5 R). American Psychiatric Pub, 2013.

[6] A. Baird et al., “Automatic classification of autistic child vocalisations: A novel database and results,” Interspeech, 2017.

[7] S. Baron-Cohen, A. M. Leslie and U. Frith, “Does the autistic child have a ”theory of mind”?” Cognition, 1985.

[8] T. Belpaeme et al., “Multimodal child-robot interaction: Building social bonds,” Journal of Human-Robot Interaction, 2012.

[9] C. M. Bishop, Pattern Recognition and Machine Learning. Springer, 2006.

[10] E. Castello Ferrer ´ et al., “Robochain: A secure data-sharing framework for human-robot interaction,” eTELEMED, 2018.

[11] F. Chollet et al., “Keras,” 2015. [Online]. Available: https: //github.com/fchollet/keras

[12] A. Chorianopoulou et al., “Engagement detection for children with autism spectrum disorder,” in IEEE ICASSP, 2017.

[13] C. E. Clabaugh, “Interactive personalization for socially assistive robots,” in IEEE HRI, 2017.

[14] M. B. Colton et al., “Toward therapist-in-the-loop assistive robotics for children with autism and specific language impairment,” Autism, 2009.

[15] D. Conti et al., “A cross-cultural study of acceptance and use of robotics by future psychology practitioners.” IEEE, 2015.

[16] T. C. Daley, “The need for cross-cultural research on the pervasive developmental disorders,” Transcultural Psychiatry, 2002.

[17] H. Elfenbein and N. Ambady, “On the universality and cultural specificity of emotion recognition: A meta-analysis,” Psychological Bulletin, 2002.

[18] P. G. Esteban et al., “How to build a supervised autonomous system for robot-enhanced therapy for children with autism spectrum disorder,” Paladyn, Journal of Behavioral Robotics, 2017.

[19] K. He, X. Zhang, S. Ren and J. Sun, “Deep residual learning for image recognition,” in IEEE CVPR, 2016.

[20] A. S. Heinsfeld et al., “Identification of autism spectrum disorder using deep learning and the abide dataset,” NeuroImage: Clinical, 2018