STATIC BODY EXPRESSION RECOGNITION WITH OPENPOSE

An Undergraduate Research Scholars Thesis

by

SICONG HUANG

Submitted to the Undergraduate Research Scholars Program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. Anxiao Jiang

May 2021

Major: Computer Science

TABLE OF CONTENTS

ABSTRACT

Static facial Expression Recognition with Openpose

Sicong Huang
Department of Computer Science and Engineering
Texas A&M University

Research Advisor: Dr. Anxiao Jiang
Department of Computer Science and Engineering
Texas A&M University

This thesis gives a reliable machine model that recognizes the action of "look down at phone" and distinguishes it from other similar actions in a given consecutive video. It first reproduces facial recognition research of dimpler. Then it moves on to introduce body action recognition and explains key factors like landmarks and Openpose used in the research. It then presents the action of "look down at phone" as the research focus and briefly mentions related works to the topic. Later on, the thesis presents methods on how facial expression is performed and quickly moves on the body expression detection techniques. For body expression detection, the thesis first explains the process with image inputs, then continues to explain the process for video samples. At the end of the methods chapter, it demonstrates how the machine model processes a 26-second video with complex actions and gives a reliable and correct estimation of the actions the video contains. The thesis later presents the results of this research and compares the estimation given by the machine model to the true answers of given samples. At last, the thesis concludes the effect and benefit of this machine model and suggests future works for this research.

DEDICATION

For Texas A&M University, who gave me the chance to flourish as an engineer.

ACKNOWLEDGMENTS

I would like to thank my research advisor, Dr. Anxiao Jiang for contributing his free time to guiding me through the whole research and teaching me not only the research method but also the critical think skill.

I would also like to give a special thanks to my friend Lida Zhang. Who spent countless time helping me out with cardinal concepts and reviewing my program and researching method. Without Dr. Jiang and Lida, I couldn't possibly finish my thesis with speed and quality. Another friend who has helped me is Shuaifang Wang. We initiated to complete the thesis together, although we decided to conduct our own research in the middle, we helped each other in many aspects both inside and outside of this project.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience. I also want to extend my gratitude to the LAUNCH staff who always remain attentive and provided splendid feedback to everyone amid the hundreds, if not thousands of applicants and their advisors.

Finally, thanks to my mother and my grandparents for their encouragement and mental supports throughout the whole learning experience.

NOMENCLATURE

adam	adaptive moment estimation
AI	Artificial Intelligence
CMU	Carnegie Mellon University
CUDA	Compute Unified Device Architecture
json	JavaScript Object Notation
MLP	Multi-layer Perception
rbf	Radial basis function
SVC	Support Vector Classifiers
SVM	Support Vector Machine
TOI	Transdermal Optical Imaging

CHAPTER I

INTRODUCTION

Facial recognition is very important, and it has been used in many aspects of life. It is a complex process that has not yet refined. Many factors can play a role during encoding, which makes the recognition not accurate enough. In addition, based on the previous research that has been done in facial recognition, the images in facial action coding system are still very limited. the research builds on the existing methods for facial recognition and focus on generating more accurate and diverse results by using deep learning with advanced facial landmarks technique and categorical model. The goal of this research is focusing on improving the accuracy and diversity of facial expression recognition with deep learning.

Just like humans who mostly recognize and determine facial expression and body actions through instincts and experiences, the machine learns them through supervised training to increase accuracy, too. Machines determine whether someone is smiling by looking at his or her lips and/or cheeks with the help of human classified facial landmarks.

Body action expression recognition is another vital field in artificial intelligence (AI). Machines that can detect body motions can understand basic actions made by humans enable computers to automate tasks of recognizing some frequent actions done by human through conventional cameras. Through the help of body landmarks, reliable and accurate machines that can detect specific actions have been used in commercial purpose and achieved high popularity in the industry. The action of "looking down at phone" is a common action that people frequently perform on daily basis. The purpose of this research is to accurately detect and distinguish this action from a given video clip.

Conventional computer vision techniques in the body expression research focus on detecting and labeling critical facial landmarks to identify features and obtain information from conventional cameras.

Definition of Landmarks

Landmarks, when used in AI research, represent a set of keypoints gathered from a human. Facial landmarks usually include keypoints representing the location of nose, eyes, eyebrows, lips, chin, etc. While body landmarks usually collect keypoints from hands, feet, arms, shoulders, neck, etc.

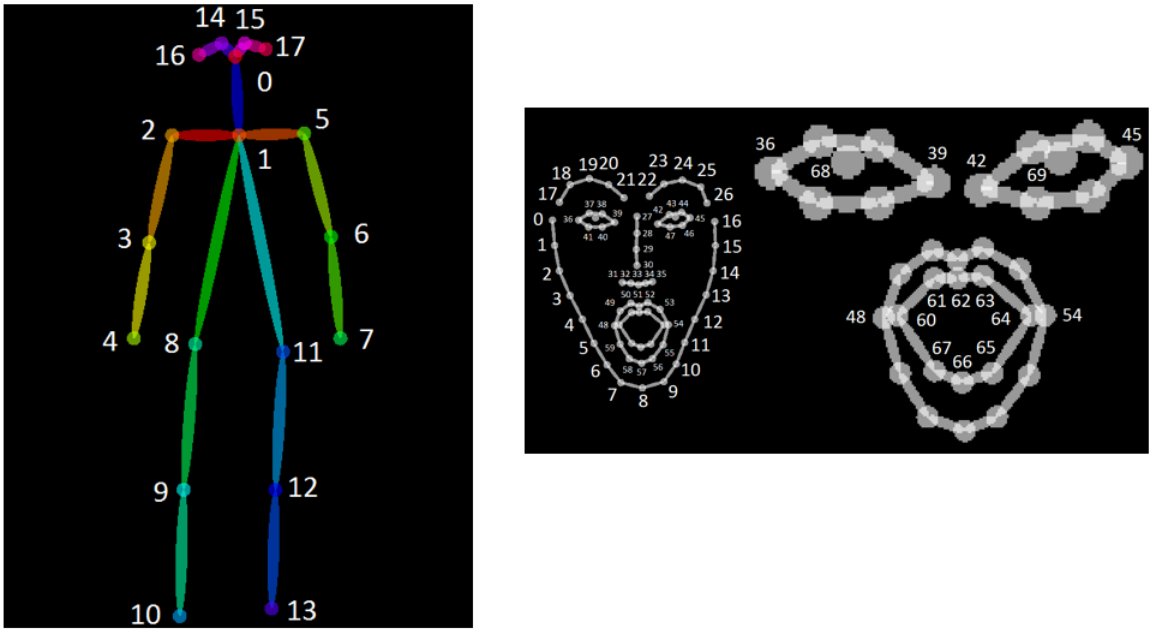

Figure 1: demonstration of facial and body landmarks

Kernel Functions in Supported Vector Machine

Kernel functions take data as input and transform it into the desired form by using complex mathematical calculations and statistical formulations [1]. In machine learning, a "kernel" is a method of using a linear classifier to solve a non-linear problem. Different kernel functions can fit deep learning features to different dimensions, finding correlations and threshold among the data samples. The kernel is fundamental to the supported vector classifier (SVC) which analyzes and classifies data according to the model and marks true fields among the whole data grid.

Body Landmarks Detection with Openpose

Openpose is a software that detects the human body, hand, facial, and foot keypoints on single images and videos[2][3]. Body keypoints include nose, neck, shoulders, elbows, wrist, knee, ankles, etc. Hand keypoints contains all the fingers, their joints, and wrist[4]. Facial keypoints include eyebrows, eyes (both upper and lower), nose, mouths, and chins. Foot keypoints contain similar results as the hand keypoints[5].

"Look down at Phone"

Constantly looking down at the phone is people's new addiction in this age of the cellphone. A survey claims that Americans on average check phones once every 12 minutes. The survey also claims that Americans repeat the same action at least 80 times every day. If the machine can automatically detect the action of "looking down at phone", it will enable more research and business relating to cellphone use. For example, schools may implement the machine to prevent the illegal use of cellphone during exams.

Related Works

Previously, action pattern recognition has been performed to testify motion sequence information using deep learning [6]. Their emphasis was on detecting motions on various temporal regions to generate a new motion representation structure.

Another related research focused on localizing actions to decompose and record temporal "key poses" in given data to achieve superior performance with high accuracy [7]. The model they developed could only detect actions but failed to classify and recognize the specific actions. This research's focus is on detecting the specific, cardinal action of "look down at phone" instead.

A similar research performed by Zhao dedicated to recognizing all actions performed in a static image [8].He and his fellows limited the scope to be any given static image and detected basic actions performed by different parts of a human body. For example, an image with the action of "pour the liquid down" would be detected as a human with actions like "head looking down", "right arm curving down", and "left hand half holding" happening simultaneously. This research's focus is instead on recognizing the macro action of "look down at phone", much like the "pour the

liquid down" action in their example.

A research that focuses on recognized cellphone use actions has been taken place[9]. But their research dedicated to recognizing driver's abnormal behavior and only tested their model on vehicle drivers using the camera mounted on the dashboard.

CHAPTER II

METHODS

A machine learning library called scikit-learn was used throughout the data analysis model because it was capable of carrying out multi-layer perceptron and support vector machines [1].

Using SVM

The SVM library used for this research was provided by scikit-learn. SVM has different estimators like Linear Support Vector Classification (LinearSVC), Nu Support Vector Classification (NuSVC), and C Support Vector Classification (SVC) [1]. SVC is chosen because data samples are non-linear and this research doesn't control the number of support vectors. To use SVC on Python, the data samples must be formatted into a 2D array (matrix) of size $N\ X\ M$ while N represents the number of samples and M represents the number of features of each sample. Because this training is supervised, the SVC also requires a 1D array of size N that represents the true answer (target value) of each sample. SVC also allows users to choose different kernels, gamma (kernel coefficient), etc. After some adjustments, this research determines that the SVC with rbf kernel and "scaled" gamma. A gamma of "scale" means the kernel coefficient equals to 1 / (**number of features** multiplies **the variance of data samples**).

After calling the SVC function, its returned SVM model can be used to fit the training samples (same structure as the previous $N\ X\ M$ data samples) to estimate the results for those answers as a 1D array (same structure as true answer array). By comparing the estimated results and true answers, the accuracy can be calculated and the trend of the estimation can be plotted.

Facial Expression Detection

For the data analysis model, the same scikit-learn library [1] was used. For the training samples, 20 static pictures with Huang's face were taken using a conventional camera on a cell phone. Then these pictures were sorted and labeled sequentially, marking the ones with dimpler with "1" and otherwise "0", storing the values to a text file as the true value for the training samples.

Because this research is supervised, the accuracy of the true values substantially determines the accuracy and use case of this research.

Proceed to the data sample localization, AI research scientist Adrian Bulat's published GitHub repository of face alignment library[10] was used to generate facial landmarks. After denoising and localizing facial landmarks from the data samples with his library, data grids for corresponding facial landmarks for further empirical analysis.

With the data grids collected previously, those coordinates are used to rationalize facial landmarks with supplied data and categorize facial behaviors with mathematical values. For example, predicting the curve of lips is a way to supervise the machine to find a correct threshold. Moving on the data model design, four different kernels functions are used for the supported vector machine approach (**Figure 2**).

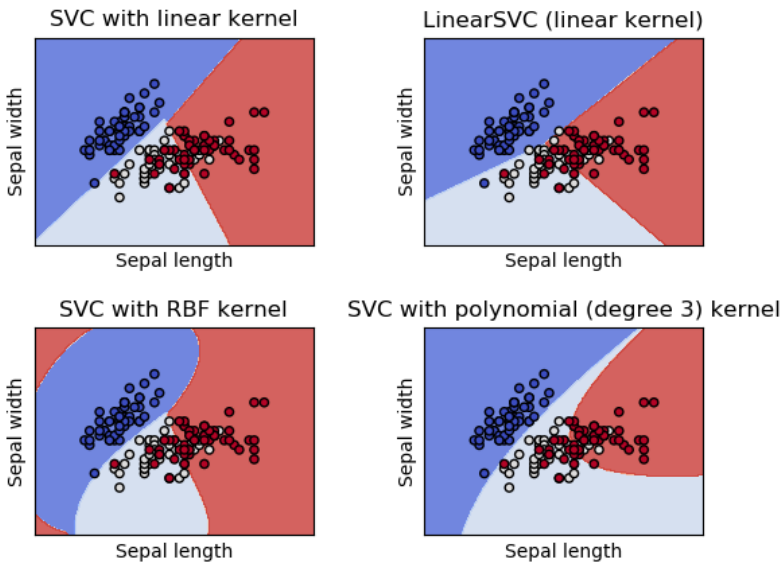

Figure 2: Different kernels can yield different estimation of the same sample using SVC [1]

Body Expression Detection with images

For body expression detection, the Openpose package provided by the Perceptual Computing Lab at CMU was used. After downloading and installing Openpose on a windows desktop, 15 static images of a person performing some actions taken by a conventional camera were used and inputted to the computer ("op_pic_#_.jpg" in the "Sicong-deep-learning/my_pics" directory). After marking them accordingly, all the images (usually denoted as samples) were processed using Openpose and only configured the software to only detect body and facial keypoints (**Figure 3**).

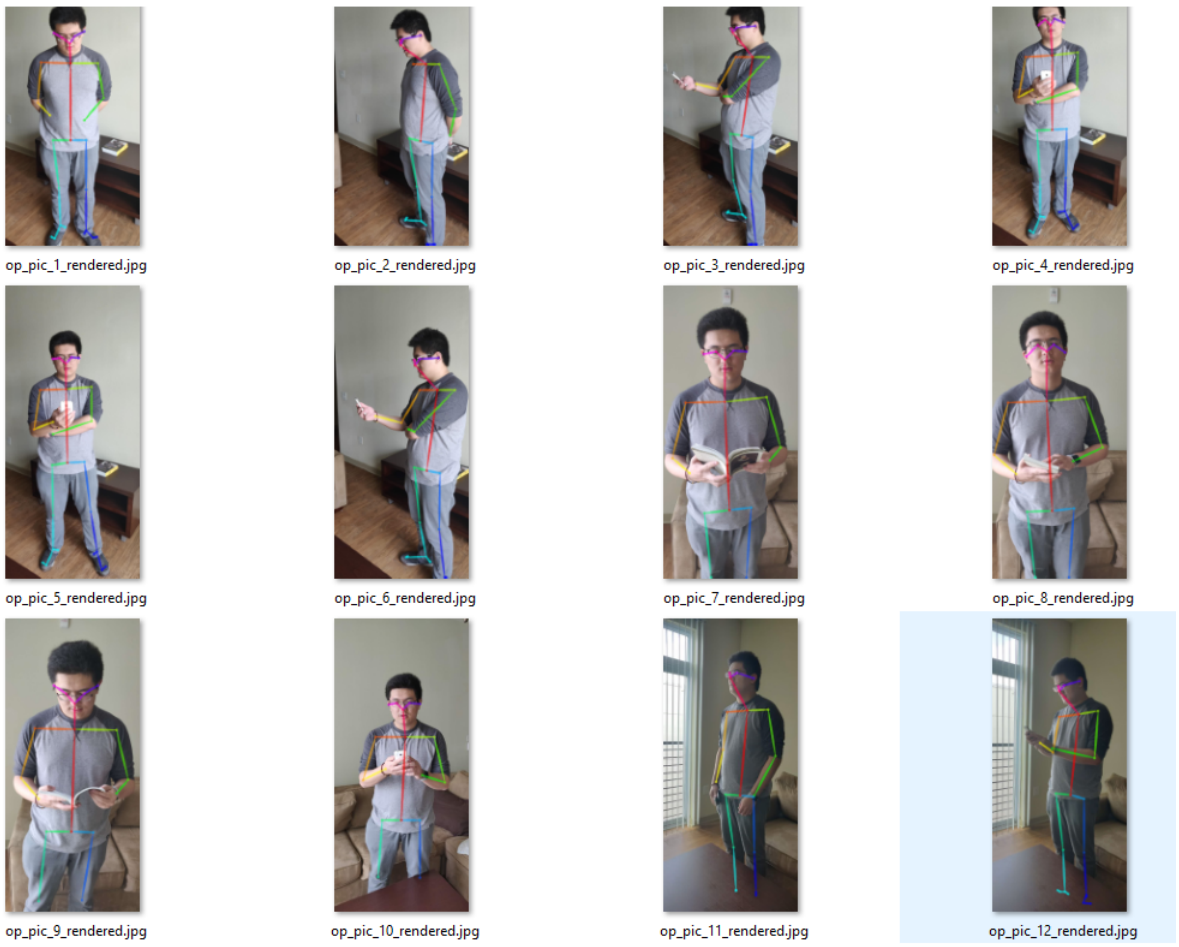

Figure 3: Rendered images with keypoints clearly marked

Openpose then produced a sequence of rendered images of jpg format with body landmarks identified body ("op_pic_#_rendered.jpg" in the "Sicong-deep-learning/my_pics/output_pic" directory) and facial keypoints and a list of json data files for corresponding samples in the same directory. This concludes the first round of data collection in this project. For data process, a csv file called "ans_all_15_images.csv" in the "Sicong-deep-learning/my_pics/json_datas" directory was used to store true answers (target values). Images with positive "looking at phone" action were denoted as 1 and otherwise as 0 [11] were stored in the csv file mentioned above. By doing so, a 1D array of size N for answers to the samples was made for future use. Another csv file called "input_all_15_images.csv" in the "Sicong-deep-learning/my_pics/json_datas" directory was used to collect and store keypoints from each json files generated above with each row representing keypoints of an image. Thus, a 2D array of size $N\ x\ M$ where N denotes as the number of samples and M denotes as the number of keypoints for each sample was created. This concludes the data processing part of the research.

After uploading all the materials to a GitHub, the research proceeded on to Python programming. With different datasets, It first inputted the "input_all_15_images.csv" to get the $N\ x\ M$ array. Because every keypoint has x label, y label, and keypoint scale and only x label and y label were needed to proceed, all the keypoint scales were removed and then pair on the x and y labels. The updated 2D array will have 2/3 of its original size and correspond to M/3 landmarks collected. There are a total of 15 landmarks and 15 samples for this step, the 2D array should always a size of *15 X 30* by the end of this step.

The next step is to create features from these keypoints. To detect whether the person in the sample is looking down at the phone, the only relevant landmarks are neck, both elbows, wrists, and eyes. After filtering, the 7 landmarks, each with x label and y label, will shorten the size of the 2D array to *15 X 14*. Using the keypoints, 2 different features were used to improve the machine's accuracy. The first feature is simple, the briefly organized raw data were dumped in. In the second feature partitioned left wrist and elbow to calculate the curve correspondingly and did the same with the right wrist elbow, and shoulder. It then generated a new 2D array with a size of *15 X 7*

array that contains curves of both arms and raw data for neck and both eyes. In the meantime, the 1D array representing the true answers of the samples remain unchanged.

With the help of PyCharm, a Python IDE, the same scikit-learn library plugin for Python to perform the dataset training with deep learning was used. Using the "train_test_split()" function provided by scikit-learn, a random 30% percent of the samples were used to the training sample and everything else was considered test samples. It has been established that SVM performs the highest accuracy within the experiment setting. To use different kernels to find out that rbf kernel fits the dataset with the highest accuracy after 100 iterations of random test & training samples, the SVM function produced a 1D array of N (size of 15) and represented the predicted results for these training samples. When compared to the true answers of the samples, the accuracy on average is 86.465%. This concludes the first portion of Body expression, the research then moved on the detecting a sequence of videos instead of individual, unrelated images.

Body Expression Detection with videos

With help from friends, 8 videos performing "looking down at phone" action were collected. To produce a more comprehensive result, the videos not only included a person acting as different background, using both hands or one hand, looking sideway and directly at the camera, and start at either true or false action. Because most conventional cameras usually record videos in MP4 format, VLC media player were used to convert them to avi format manually. To reduce the workload while keeping the integrity of the sample, "frame_step" flag was used to only detect body landmarks every 5 frames and treat every video as an individual sequence of consecutive images so that they could be proceeded using similar data processing techniques mentioned above but storing all of the data and materials in the "Sicong-Deep-Learning/my_videos" (**Figure 4**).

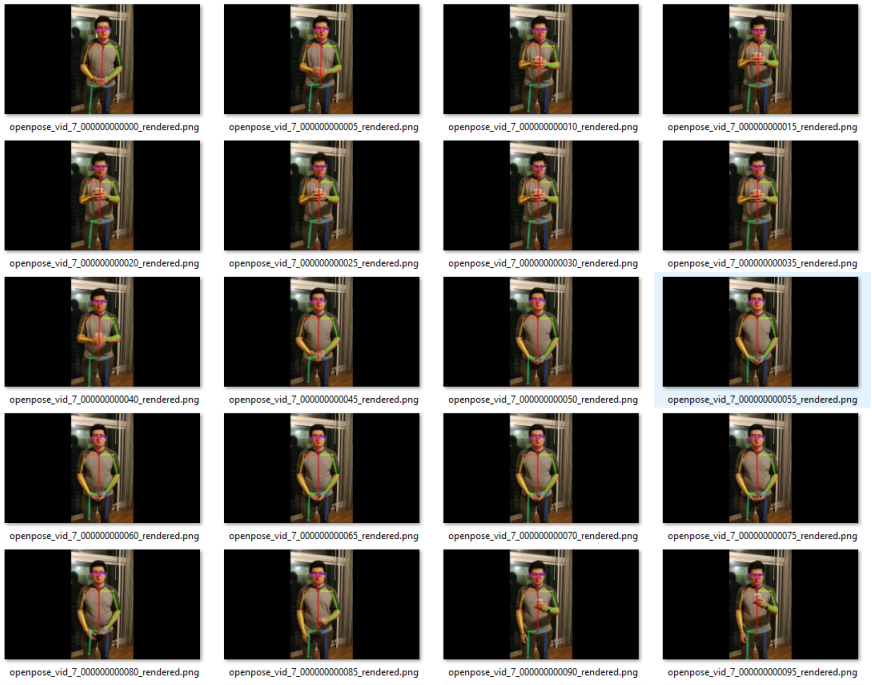

Figure 4: Rendered consecutive frames with keypoints clearly marked, to better visualize the video

Later, the 8 sets of samples were divided into two categories, “looking at the camera” and “looking sideways”. Using SVM with the same kernel, it achieves a much higher accuracy with “looking at the camera” than otherwise and received the permission from the advisor to proceed.

The final training sample was a long, consecutive video where the person performed various actions in front of the camera. It was a 26-second video where the person first looked at the phone, put on a hat, drank water, and at least looked at the phone again. When processing the video, it was decided to only record and generate every 10 frames due to the extensive size. After viewing the consecutive images, no visual discord was detected so the research proceeded to the next step. Using the same technique above, a new sample marked as “avi_9” with 79 frames of keypoints was generated. After processing and filtering the 2D array of size *79 X 75* with the same technique above, the feature could accurately distinguish among “looking down at phone” and other actions.

CHAPTER III

RESULTS

For facial expression recognition, the research successfully reproduced an accurate dimpler detection model and yields an accuracy of over 88%. For body expression detection, the result of single images maintains a rate of 86.465% accuracy when using the SVM with rbf kernel provided by scikit-learn library. SVM yields reliable results by focusing on the local minimum. Although MLP was considered, SVM fits the data better because as an individual undergraduate student, the lack of funding and resource limited the ability to collect extensive datasets and samples for the research to take the advantage of the more accurate and reliable model provided by MLP. Although the adam function [12] does indeed leveraged its adaptive nature in seeking correlations among the dataset with the relatively small input sample size, MLP still fails to provide a more reliable estimation of samples and were thus disregarded when moving on.

After collecting short videos for the next step of the research, using SVM with rbf kernel consistently provided the best estimation of the models among other kernels and providing a result that correlated to the results above (when samples were images). However, one of the flaws in this stage was the inconsistency when the person in the camera was looking sideways. Because body landmarks have utterly different positions and movement curves depending on which way the person was facing. It was conceivable to suspect that the deep learning model overfitted the samples of people looking directly at the camera and neglected the instances otherwise. Despite this incident, the overall outputs consistently correlated with true samples when people looking directly at the camera and only had a few spikes on where the people were looking at the data.

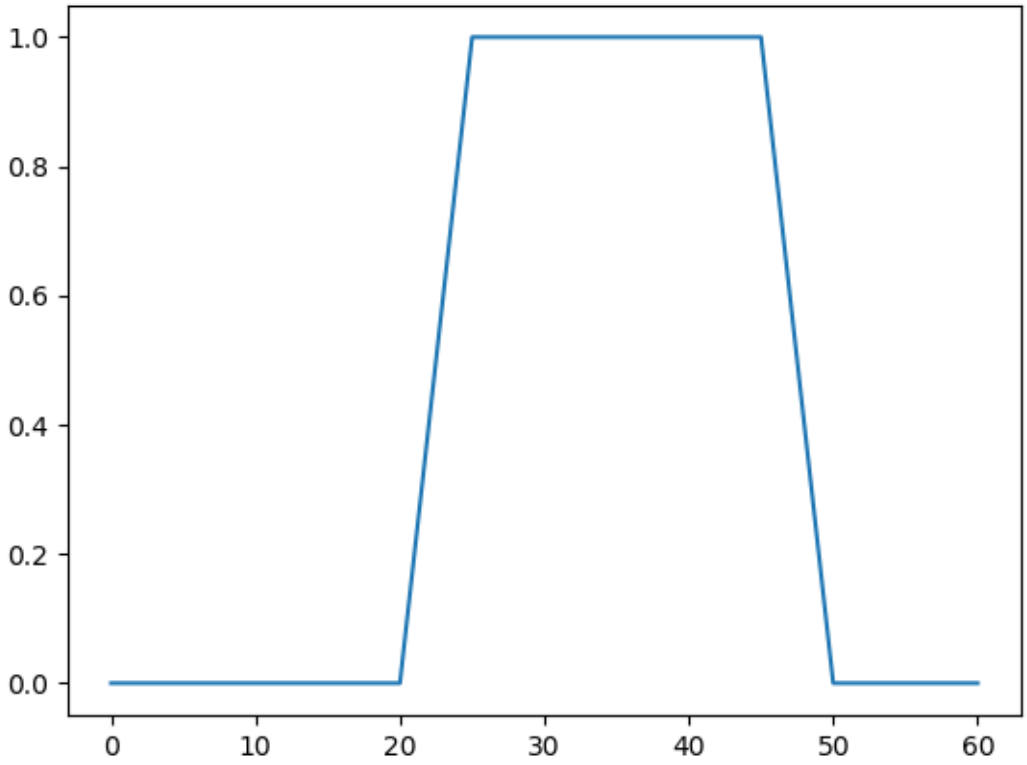

Figure 5: Estimated result of video sample 1

For the above graphs, the x-axis represents the number of frames and the y-axis represents the estimated result of that frame. For example, at frame 0 (the first frame), the model predicted the image didn't contain the action of "look down at phone" and represented its finding by setting the y value to 0 (**Figure 5**).

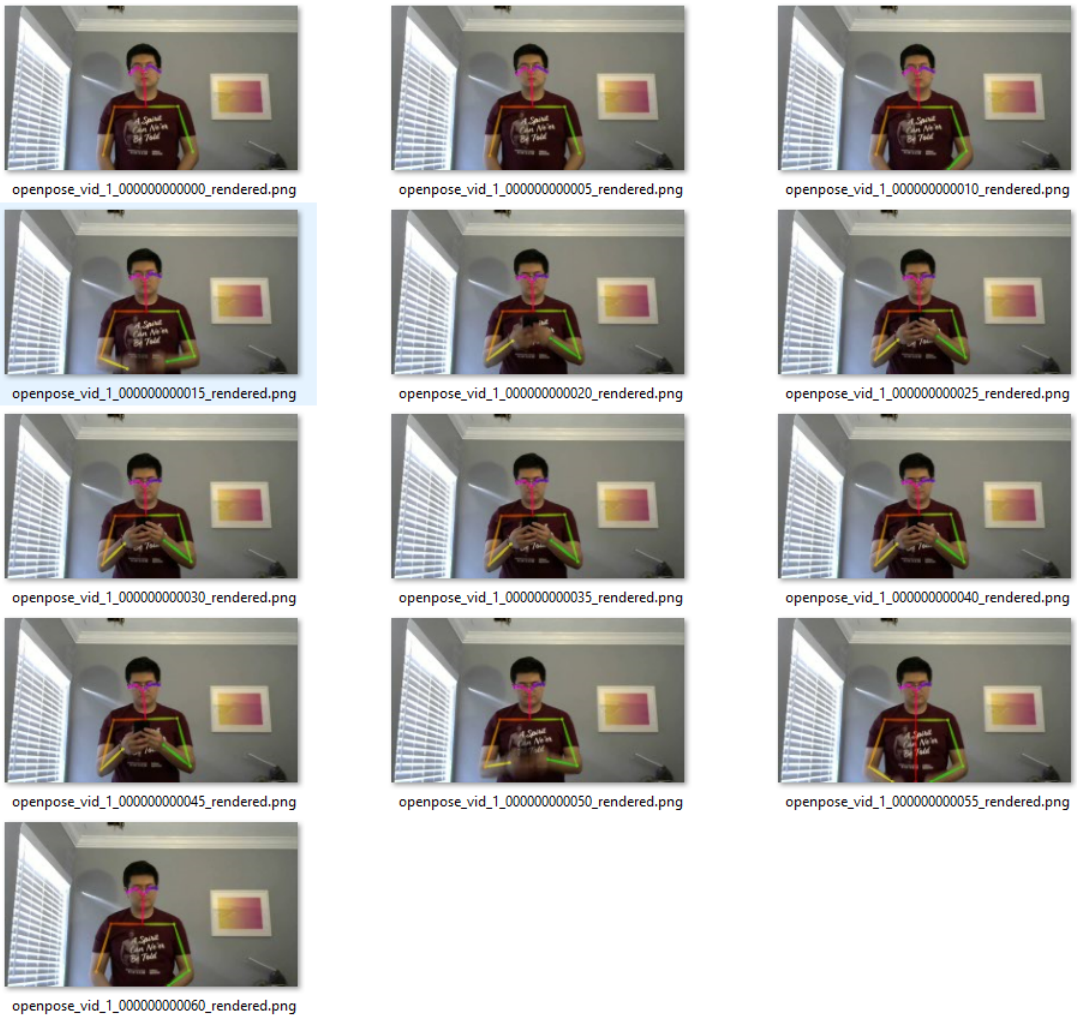

Figure 6: Consecutive frames of sample video 1, as compared to data graph above, the result is very accurate

As we see from the actual frames above (**Figure 6**), the spikes are neglectable because the model only makes mistakes at transitioning frames, which can be tricky to humans as well. When viewing the spikes, although those spikes were lowering the overall score of the dataset, the data could still consistently detect the start and end frame of the looking down at phone action. With some additive processing, the model gives a reliable interval to represent which portion of the samples have positive action.

Meanwhile, for the final sample (video 9) , a comprehensive representation of the whole progress on body expression detection is presented below (**Figure 7**).

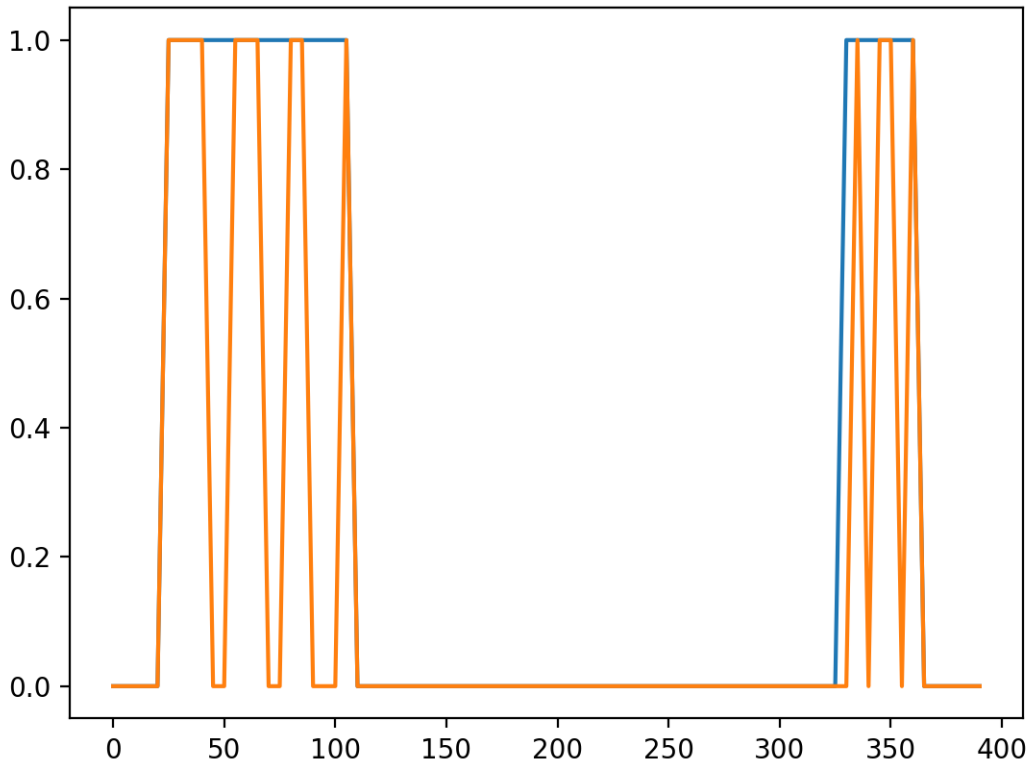

Figure 7: Blue line represents true answers and orange line represents estimated answers

As mentioned above, the person started by looking down at the phone. He then put on a hat and drank water from a water bottle. At the end of the video, he looked down at the phone again. Although the spikes mentioned before still exist in the image, this accuracy of this model is shockingly high. Because of it, an interval of "looking down at phone" action can be easily extracted with high confidence.

CHAPTER IV

CONCLUSION

The research builds upon preexist landmark detection technology and deep learning techniques to detect one of the most prevalent contemporary actions, looking down at the phone. At the end of the research, it successfully trained a reliable machine model that can accurately recognize the action of "look down at phone" and distinguish this action from other similar actions like "put on a hat", and "drink water from a bottle".

Facial and Body Landmarks

The use of facial and body landmarks can substantially improve the accuracy of the deep learning model. This is because those classified points are pivots to define each facial expression and body actions. Throughout this research, the research confirms that 7 landmarks are enough to generate a reliable and efficient deep learning model to detect most conventional motions.

Benefit to the Society

Because the model can accurately detect an interval of frames inside a video. This model, after being modified as an API, can be easily used for places where the use of cellphones is prohibited. A user story example is provided below. *As law enforcement, this technology can help me track drivers who look at their phones while waiting for traffic lights so that I can enforce the local law of "no texting while driving" without dispatching officers at every block of the roads*.

Future Works

The result of this research meets the expectation within the time frame. Although the accuracy is not the state of the art, the research result is acceptable. After enhancing the data accordingly, the interval provided in the result, the interval evaluated by the end of the research provided a confident estimation of the action with given samples.

To increase the integrity of the current model, it's critical to collect and include larger samples because everyone performs the same actions slightly differently. Another flaw of the

current model is its low accuracy against samples with people performing the action not directly to the camera. To resolve this issue, a more diverse sample might be needed but more importantly a new feature that compensates the impact of an object not directly facing the camera. The method presented here allows fellow scholars to replicate or perform new body action recognition like "sweep the floor" and "drink water from a bottle".

REFERENCES

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[2] S.-E. Wei, V. Ramakrishna, T. Kanade, and Y. Sheikh, "Convolutional pose machines," in *CVPR*, 2016.

[3] Z. Cao, T. Simon, S.-E. Wei, and Y. Sheikh, "Realtime multi-person 2d pose estimation using part affinity fields," in *CVPR*, 2017.

[4] T. Simon, H. Joo, I. Matthews, and Y. Sheikh, "Hand keypoint detection in single images using multiview bootstrapping," in *CVPR*, 2017.

[5] Z. Cao, G. Hidalgo, T. Simon, S.-E. Wei, and Y. Sheikh, "OpenPose: realtime multi-person 2D pose estimation using Part Affinity Fields," in *arXiv preprint arXiv:1812.08008*, 2018.

[6] E. P. Ijjina and K. M. Chalavadi, "Human action recognition in rgb-d videos using motion sequence information and deep learning," *Pattern Recognition*, vol. 72, pp. 504–516, 2017.

[7] L. Wang, Y. Qiao, and X. Tang, "Video action detection with relational dynamic-poselets," in *European conference on computer vision*, pp. 565–580, Springer, 2014.

[8] Z. Zhao, H. Ma, and S. You, "Single image action recognition using semantic body part actions," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[9] X. Zhang, N. Zheng, F. Wang, and Y. He, "Visual recognition of driver hand-held cell phone use based on hidden crf," in *Proceedings of 2011 IEEE International Conference on Vehicular Electronics and Safety*, pp. 248–251, 2011.

[10] A. Bulat and G. Tzimiropoulos, "How far are we from solving the 2d & 3d face alignment problem? (and a dataset of 230,000 3d facial landmarks)," in *International Conference on Computer Vision*, 2017.

[11] J. Schmidhuber, "Reinforcement learning upside down: Don't predict rewards – just map them to actions," 2019.

[12] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

APPENDIX

A demo of how Openpose detects and records landmarks https://youtu.be/JDaMm2D2N4w

Full playlist of videos samples used

https://www.youtube.com/playlist?list=PLIgse6IBhew61dIVYYnX-Yo5n6Ng25wmX

GitHub repository of this research https://github.com/Innoversa/Sicong-deep-learning

Sicong_1.py

This code is used to extract facial landmarks using face-alignment library [10]

```
import face_alignment
import matplotlib
# matplotlib.use('Agg') // uncomment if plotting through remote
   machine
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from skimage import io
import collections
import csv
import numpy

def get_img_data(path):
    # Run the 3D face alignment on a test image, without CUDA.
    fa = face_alignment.FaceAlignment(face_alignment.
       LandmarksType._3D, device='cpu', flip_input=True)
    input_img = io.imread(path)
    preds = fa.get_landmarks(input_img)[-1]
    # 2D-Plot
```

```
    plot_style = dict(marker='o',
                      markersize=4,
                      linestyle='-',
                      lw=2)

    pred_type = collections.namedtuple('prediction_type', ['slice
        ', 'color'])
    pred_types = {'lips': pred_type(slice(54, 60), (0.596, 0.875,
         0.541, 0.3)),
                  'eyebrow1': pred_type(slice(17, 22), (1.0,
                     0.498, 0.055, 0.4)),
                  'eyebrow2': pred_type(slice(22, 27), (1.0,
                     0.498, 0.055, 0.4))}
    data = numpy.concatenate((preds[pred_types['lips'].slice,
        0:2], preds[pred_types['eyebrow1'].slice, 0:2], preds[
        pred_types['eyebrow2'].slice, 0:2]))
    return data


with open('data.csv', mode='w') as data_file:
    data_writer = csv.writer(data_file, delimiter=',', quotechar=
        '"', quoting=csv.QUOTE_MINIMAL)
    # data_writer.writerow(['x_value', 'y_value'])
    for i in range(17):
        if i < 10:
            path = 'pics/dimpler_0'+str(i)+'.png'
        else:
            path = 'pics/dimpler_'+str(i)+'.png'
```

```
print(path)
data = get_img_data(path)
data_writer.writerows(data)
```

Sicong_2.py

This code is used to train features for the facial recognition research [1]

```
import csv
import numpy as np
from sklearn import svm
from sklearn.svm import LinearSVC
import sklearn
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
from sklearn import datasets
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import LinearSVC
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import (brier_score_loss, precision_score,
    recall_score, f1_score)
from sklearn.calibration import CalibratedClassifierCV,
    calibration_curve
from sklearn.model_selection import train_test_split


def plot_calibration_curve(est, name, fig_index):
    """Plot calibration curve for est w/o and with calibration.
       """
    # Calibrated with isotonic calibration
    isotonic = CalibratedClassifierCV(est, cv=2, method='isotonic
       ')
```

```
# Calibrated with sigmoid calibration
sigmoid = CalibratedClassifierCV(est, cv=2, method='sigmoid')

# Logistic regression with no calibration as baseline
lr = LogisticRegression(C=1.)

fig = plt.figure(fig_index, figsize=(10, 10))
ax1 = plt.subplot2grid((3, 1), (0, 0), rowspan=2)
ax2 = plt.subplot2grid((3, 1), (2, 0))

ax1.plot([0, 1], [0, 1], "k:", label="Perfectly␣calibrated")
for clf, name in [(lr, 'Logistic'),
                  (est, name),
                  (isotonic, name + '␣+␣Isotonic'),
                  (sigmoid, name + '␣+␣Sigmoid')]:
    clf.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    if hasattr(clf, "predict_proba"):
        prob_pos = clf.predict_proba(x_test)[:, 1]
    else:  # use decision function
        prob_pos = clf.decision_function(x_test)
        prob_pos = \
            (prob_pos - prob_pos.min()) / (prob_pos.max() -
              prob_pos.min())

    clf_score = brier_score_loss(y_test, prob_pos, pos_label=
      Y.max())
```

```
        print("%s:" % name)
        print("\tBrier: %1.3f" % (clf_score))
        print("\tPrecision: %1.3f" % precision_score(y_test,
            y_pred))
        print("\tRecall: %1.3f" % recall_score(y_test, y_pred))
        # print("\tF1: %1.3f\n" % f1_score(y_test, y_pred))

        fraction_of_positives, mean_predicted_value = \
            calibration_curve(y_test, prob_pos, n_bins=10)

        ax1.plot(mean_predicted_value, fraction_of_positives, "s-
            ",
                 label="%s (%1.3f)" % (name, clf_score))

        ax2.hist(prob_pos, range=(0, 1), bins=10, label=name,
                 histtype="step", lw=2)

    ax1.set_ylabel("Fraction of positives")
    ax1.set_ylim([-0.05, 1.05])
    ax1.legend(loc="lower right")
    ax1.set_title('Calibration plots  (reliability curve)')

    ax2.set_xlabel("Mean predicted value")
    ax2.set_ylabel("Count")
    ax2.legend(loc="upper center", ncol=2)

    plt.tight_layout()
```

```
def featureAll(input1):
    print(input1)
    # input("hello")
    output = []
    output += (feature1(input1))
    # output += (feature2(input1))
    output += (feature3(input1))
    output += feature4(input1[7::])
    return output


def feature1(input1):
    # feature regarding lips
    output = [0, 1, 2, 3]
    output[0] = (input1[0][0] - input1[3][0])
    output[1] = (input1[0][1] - input1[3][1])
    output[2] = (input1[6][0] - input1[3][0])
    output[3] = (input1[6][1] - input1[3][1])
    # output_val = output[0] + output[1] * 10 + output[2] * 100 +
        output[3] * 1000
    return [output[1]-output[0], output[3]-output[2]]
    # return output
```

```
def feature2(input1):
    # feature regarding lips
    output = [0, 1, 2, 3, 4, 5, 6]
    output[0] = input1[0][1]
    output[1] = input1[1][1]
    output[2] = input1[2][1]
    output[3] = input1[3][1]
    output[4] = input1[4][1]
    output[5] = input1[5][1]
    output[6] = input1[6][1]
    return output


def feature3(input1):
    # feature regarding lips
    output = [0, 1, 2, 3, 4, 5]
    output[0] = input1[0][0]
    output[1] = input1[0][1]
    output[2] = input1[3][0]
    output[3] = input1[3][1]
    output[4] = input1[6][0]
    output[5] = input1[6][1]
    return output


def feature4(input2):
    output = []
```

```
    for e in range(len(input2)):
        output.append(input2[e][0])
    return output


with open('data.csv', mode='r') as data_file:
    data_reader = csv.reader(data_file, delimiter=',', quotechar=
       '\'', quoting=csv.QUOTE_NONNUMERIC)
    outp = []
    for each in data_reader:
        outp.append(each)

with open('answer.csv', mode='r') as data_file:
    data_reader = csv.reader(data_file, quoting=csv.
       QUOTE_NONNUMERIC)
    ans = []
    for each in data_reader:
        ans = each
    ans = list(map(int, ans))

with open('fake_smile.csv', mode='r') as data_file:
    data_reader = csv.reader(data_file, quoting=csv.
       QUOTE_NONNUMERIC)
    fake = []
    for each in data_reader:
        fake = each
    fake = list(map(int, fake))
```

```
with open('happy_smile.csv', mode='r') as data_file:
    data_reader = csv.reader(data_file, quoting=csv.
       QUOTE_NONNUMERIC)
    happy = []
    for each in data_reader:
        happy = each
    happy = list(map(int, happy))
    # print(ans)


f1 = []
for a in range(17):
    f1.append(featureAll(outp[a * 16:a * 16 + 16]))
    # f1.append(featureAll(outp))
    # f1.append(feature3(outp))
print('printing', f1)
print('len = ', len(f1))
# print(ans)
X = np.array(f1)
# print(X)
Y = np.array(ans)
# print(Y)
acc = 0
f1_score = 0
score = 0
score2 = 0
for i in range(1000):
```

```
    x_train, x_test, y_train, y_test = sklearn.model_selection.
        train_test_split(X, Y)
    # clf = svm.SVC(degree=3, gamma='scale', kernel='poly',
        decision_function_shape='ovr')
    clf = svm.LinearSVC(random_state=0, tol=1e-5)
    # clf2 = MLPClassifier(solver='adam', alpha=1e-5,
        hidden_layer_sizes=(5, 2), random_state=1)
    clf.fit(x_train, y_train)
    # clf2.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    # y_pred2 = clf2.predict(x_test)
    # score2 += MLPClassifier.score(clf2, X, Y)
    score += svm.SVC.score(clf, X, Y)
    acc = acc + sklearn.metrics.accuracy_score(y_test, y_pred)
    # print(sklearn.metrics.accuracy_score(y_test, y_pred))
    # f1_score = f1_score + sklearn.metrics.f1_score(y_test,
        y_pred, average='binary')
print(acc / 1000)
print(score / 10)
```

Sicong_4.py

Sicong_3.py was a refactored version of Sicong_2.py for documentation purpose. For Sicong_4.py, this code is used to train features for the body expression recognition research with images [1]

```
import csv
import numpy as np
from sklearn import svm
import sklearn

def feature_look_down_1(input_pts, input_index):
    output_pts = []
    output_pt = []
    for each_r in input_pts:
        for index in input_index:
            output_pt.append(each_r[index])
        output_pts.append(output_pt)
        output_pt = []
    # print(len(output_pts), len(output_pts[0]))
    return output_pts


# start of data processing part
with open('my_pics/json_datas/ans_all_15_images.csv', mode = 'r')
    as data_file:
    data_reader = csv.reader(data_file, delimiter=',', quotechar=
       '\'', quoting=csv.QUOTE_NONNUMERIC)
    ans_data = []
    for each in data_reader:
```

```
            ans_data = each
        ans_data = list(map(int, ans_data))
with open('my_pics/json_datas/input_all_15_images.csv', mode='r')
    as data_file:
    data_reader = csv.reader(data_file, delimiter=',', quotechar=
       '\'', quoting=csv.QUOTE_NONNUMERIC)
    all_pts = []
    for each in data_reader:
        # print(each)
        all_pts.append(each)
# print('output is ', all_pts) # all points is a 2D array with 15
    rows and 75 columns
# this modifies the list into a 15 rows and 50 columns 2D array
   that has only x and y coordinates
for each in all_pts:
    for pts in each:
        if pts <= 1:
            each.remove(pts)
# data I need is neck (1), Relbow(3), Rwrist(4), Lelbow(6),
   Lwrist(7), Reye(15), Leye(16)
need_features = [1, 3, 4, 6, 7, 15, 16]
need_idx = []
for each in need_features:
    need_idx.append(each * 2)
    need_idx.append(each * 2 + 1)
# print(need_pts) # this gets the index needed for the research
feature_1 = feature_look_down_1(all_pts, need_idx)
```

```
print(len(feature_1))
print(len(ans_data))

# start of sklearn part
X = np.array(feature_1)
Y = np.array(ans_data)

acc = 0
f1_score = 0
score = 0
score2 = 0
for i in range(1000):
    x_train, x_test, y_train, y_test = sklearn.model_selection.
        train_test_split(X, Y)
    clf = svm.SVC(degree=3, gamma='scale', kernel='rbf',
        decision_function_shape='ovo')
    # clf = svm.LinearSVC(random_state=0, tol=1e-5)
    # clf2 = sklearn.MLPClassifier(solver='adam', alpha=1e-5,
        hidden_layer_sizes=(5, 2), random_state=1)
    clf.fit(x_train, y_train)
    # clf2.fit(x_train, y_train)
    y_pred = clf.predict(x_test)
    # y_pred2 = clf2.predict(x_test)
    # score2 += MLPClassifier.score(clf2, X, Y)
    score += svm.SVC.score(clf, X, Y)
    acc = acc + sklearn.metrics.accuracy_score(y_test, y_pred)
    # print(sklearn.metrics.accuracy_score(y_test, y_pred))
```

```
    # f1_score = f1_score + sklearn.metrics.f1_score(y_test,
       y_pred, average='binary')
# print(f1_score / 1000)
print(acc / 1000)
print(score / 10)
# print(score2 / 1000)
```

Sicong_5.py

For Sicong_5.py, this code is used to train features for the body expression recognition research with videos [1]

```
import csv
import numpy as np
from sklearn import svm
import sklearn
import json



def feature_look_down_1(input_pts, input_index):
    output_pts = []
    output_pt = []
    for each_r in input_pts:
        for index in input_index:
            output_pt.append(each_r[index])
        output_pts.append(output_pt)
        output_pt = []
    # print(len(output_pts), len(output_pts[0]))
    return output_pts


def calc_accuracy(X, Y):
    acc = 0
    f1_score = 0
    score = 0
    score2 = 0
    for i in range(1000):
```

```
        x_train, x_test, y_train, y_test = sklearn.
            model_selection.train_test_split(X, Y)
        clf = svm.SVC(gamma='auto', kernel='rbf')
        clf.fit(x_train, y_train)
        y_pred = clf.predict(x_test)
        score += svm.SVC.score(clf, X, Y)
        acc = acc + sklearn.metrics.accuracy_score(y_test, y_pred
            )
    print(acc / 1000)
    print(score / 10)
    return clf


# start of data processing part
def input_data(in_dig):
    pts_var = 'my_videos/output/avi_'+str(in_dig)+'/avi_'+str(
        in_dig)+'_pts.csv'
    ans_var = 'my_videos/output/avi_'+str(in_dig)+'/avi_'+str(
        in_dig)+'_ans.csv'
    with open(pts_var, mode='r') as data_file:
        data_reader = csv.reader(data_file, delimiter=',',
            quotechar='\'', quoting=csv.QUOTE_NONNUMERIC)
        all_pts = []
        for each in data_reader:
            # print(each)
            all_pts.append(each)
    # print(len(all_pts), len(all_pts[0])) all_pts is a 13 by 75
```

```
        matrix while 13 is the time axis
    with open(ans_var, mode='r') as data_file:
        data_reader = csv.reader(data_file, delimiter=',',
            quotechar='\'', quoting=csv.QUOTE_NONNUMERIC)
        ans_data = []
        for each in data_reader:
            ans_data = each
        ans_data = list(map(int, ans_data))
    # print(len(ans_data))
    return all_pts, ans_data


#processing the pts
def do_deep_learning(in_dig):
    all_pts, ans_data = input_data(in_dig)
    # this modifies the list into a 15 rows and 50 columns 2D
        array that has only x and y coordinates
    for each in all_pts:
        for pts in each:
            if pts <= 1:
                each.remove(pts)
    # data I need is neck (1), Relbow(3), Rwrist(4), Lelbow(6),
        Lwrist(7), Reye(15), Leye(16)
    need_features = [1, 3, 4, 6, 7, 15, 16]
    need_idx = []
    for each in need_features:
        need_idx.append(each * 2)
        need_idx.append(each * 2 + 1)
```

```
# print(need_idx)
feature_1 = feature_look_down_1(all_pts, need_idx)
# print(len(feature_1))
# print(len(ans_data))

# start of sklearn part
X = np.array(feature_1)
Y = np.array(ans_data)
clf = calc_accuracy(X, Y)
# dec_func_ans = clf.decision_function(X)
dec_func_ans = clf.predict(X)
dec_func_ans = dec_func_ans.tolist()
print(dec_func_ans)
# for each in dec_func_ans:
#     if each < 0:
#         each = 1 + each
# print (dec_func_ans)
# initializing json output for Dr. Jiang's required format
dict_to_json = {}
dict_list = []
for i in range(len(dec_func_ans)):
    dict_list.append([i * 5, dec_func_ans[i]])
dict_to_json['head_down'] = dict_list
print(dict_to_json)
out_var = 'my_videos/output/avi_'+str(in_dig)+'/avi_'+str(
   in_dig)+'_json_out.json'
with open(out_var, 'w+') as outfile:
```

```
        json.dump(dict_to_json, outfile)
    print('\n\n end of ', in_dig)
# do_deep_learning(7)
for i in range(8):
    do_deep_learning(i+1)
```

slave.py

For slave.py, this code is used to plot the trends.

```
import matplotlib.pyplot as plt
import json

def input_data(in_dig):
    json_var = 'my_videos/output/avi_'+str(in_dig)+'/avi_'+str(
        in_dig)+'_json_out.json'
    with open(json_var, 'r') as data_reader:
        asd = json.loads(data_reader.read())
    return asd['head␣down']

avi = []
for i in range(8):
    avi.append(input_data(i+1))

for i in range(len(avi)):
    X = []
    Y = []
    for each in avi[i]:
        X.append(each[0])
        Y.append(each[1])
    plt.plot(X, Y)
    # plt.show()
    plt.savefig(fname='my_videos/output/fig_trend_of_avi_'+str(i
        +1)+'.PNG')
    plt.close()
```