CUROP - Face Tracker

Hristo Georgiev

2015-07-31

1 Initial task

The task was to create and automate the process of classification of Emotions and Auction Units in the future.

1.1 Splitting of the steps

In the beginning we started with only CK+ database and used their absolute addresses from the vector as a data for svm. As a results the accuracy was 96%. We found out that this method was wrong because of the connection and the dependency of the images to be the same resolution we couldn't even try to test and confirm the accuracy. The next step was when we introduced the concept of 'normalization' and as a result we could say that the svm would be able to work with different types of images. The facetracker was giving as 66 out of the possible 68 face feature points and we decided to use the distance between the eyes as normalization value, because this distance stays the same going through all the emotions. A problem which could occur is when the face is turned away from the camera, reducing the distance between the eyes drastically and as a result getting wrong prediction.

1.1.1 classification.py

This program is responsible of splitting the CK+ and KDEF database in different categories. There are images which are without emotions and we take them as 'natural' but we also take EVERY first frame of the data we have. Since the frames aways start from natural position and then go to the emotion. This could be Auction Units or Emotions. It has hardcoded paths to the databases so please be careful.

```
#!/usr/bin/env python3
import shutil
import os, sys

ck = {
    1 : "Angry",
    2 : "Contempt",
    3 : "Disgust",
    4 : "Fear",
    5 : "Happy",
    6 : "Sadness",
    7 : "Surprise"
}

data = {}
data["Natural"] = []
everything = []

for sequence in os.listdir("../Emotion"):
    for episode in os.listdir("../Emotion/" + sequence):
        natural = True
        for index in os.listdir("../Emotion/" + sequence + "/" + episode):
            index = index[:index.index("_emotion")]
            natural = False
            for i, index_content in enumerate(open("../Emotion/" + sequence + "/" +
                episode + "/" + index + "_emotion.txt")):
                emotion = ck[int(float(index_content.strip()))]
                if not emotion in data: data[emotion] = []
                os.system("cp ../cohn-kanade-images/" + sequence + "/" + episode + "/" +
                    index + ".png ; ./face_tracker " + index + ".png ; mv " + index + "*
                    ../cohn-kanade-images/" + sequence + "/" + episode + "/")
                data[emotion].append("../cohn-kanade-images/" + sequence + "/" + episode +
                    "/" + index + ".vector")
                data[emotion].append("../cohn-kanade-images/" + sequence + "/" + episode +
                    "/" + index + "_mirror.vector")
            everything.append("../cohn-kanade-images/" + sequence + "/" + episode + "/" +
                index + ".vector")
```

```
            everything.append("../cohn-kanade-images/" + sequence + "/" + episode + "/" +
                index + "_mirror.vector")
        if natural:
            for index in os.listdir("../cohn-kanade-images/" + sequence + "/" + episode):
                if ".png" in index:
                    os.system("cp ../cohn-kanade-images/" + sequence + "/" + episode + "/"
                        + index + " . ; ./face_tracker " + index + " ; mv " + index[:index
                        .index(".")] + "* ../cohn-kanade-images/" + sequence + "/" +
                        episode + "/")
                    data["Natural"].append("../cohn-kanade-images/" + sequence + "/" +
                        episode + "/" + index[:index.index(".")] + ".vector")
                    everything.append("../cohn-kanade-images/" + sequence + "/" + episode
                        + "/" + index[:index.index(".")] + "_mirror.vector")
        else:
            index = sorted(os.listdir("../cohn-kanade-images/" + sequence + "/" + episode)
                )[0]
            os.system("cp ../cohn-kanade-images/" + sequence + "/" + episode + "/" + index
                + " . ; ./face_tracker " + index + " ; mv " + index[:index.index(".")] + "
                * ../cohn-kanade-images/" + sequence + "/" + episode + "/")
            data["Natural"].append("../cohn-kanade-images/" + sequence + "/" + episode + "
                /" + index[:index.index(".")] + ".vector")
            everything.append("../cohn-kanade-images/" + sequence + "/" + episode + "/" +
                index[:index.index(".")] + "_mirror.vector")

for sequence in os.listdir("../KDEF"):
    for episode in os.listdir("../KDEF/" + sequence):
        if episode[len(episode)-3:] == "JPG" and episode[6] == "S":
            os.system("cp ../KDEF/" + sequence + "/" + episode + " . ; ./face_tracker " +
                episode + " ; mv " + episode[:episode.index(".")] + "* ../KDEF/" + sequence
                + "/")

            if episode[4:6] == "AF":
                data["Fear"].append("../KDEF/" + sequence + "/" + episode.replace("JPG", "
                    vector"))
                data["Fear"].append("../KDEF/" + sequence + "/" + episode[:episode.index("
                    .")] + "_mirror.vector")
            elif episode[4:6] == "AN":
                data["Angry"].append("../KDEF/" + sequence + "/" + episode.replace("JPG",
                    "vector"))
                data["Angry"].append("../KDEF/" + sequence + "/" + episode[:episode.index(
                    ".")] + "_mirror.vector")
            elif episode[4:6] == "DI":
                data["Disgust"].append("../KDEF/" + sequence + "/" + episode.replace("JPG"
                    , "vector"))
                data["Disgust"].append("../KDEF/" + sequence + "/" + episode[:episode.
                    index(".")] + "_mirror.vector")
            elif episode[4:6] == "HA":
                data["Happy"].append("../KDEF/" + sequence + "/" + episode.replace("JPG",
                    "vector"))
                data["Happy"].append("../KDEF/" + sequence + "/" + episode[:episode.index(
                    ".")] + "_mirror.vector")
            elif episode[4:6] == "NE":
                data["Natural"].append("../KDEF/" + sequence + "/" + episode.replace("JPG"
                    , "vector"))
                data["Natural"].append("../KDEF/" + sequence + "/" + episode[:episode.
                    index(".")] + "_mirror.vector")
            elif episode[4:6] == "SA":
                data["Sadness"].append("../KDEF/" + sequence + "/" + episode.replace("JPG"
                    , "vector"))
```

```
                data["Sadness"].append("../KDEF/" + sequence + "/" + episode[:episode.
                    index(".")] + "_mirror.vector")
            elif episode[4:6] == "SU":
                data["Surprise"].append("../KDEF/" + sequence + "/" + episode.replace("JPG
                    ", "vector"))
                data["Surprise"].append("../KDEF/" + sequence + "/" + episode[:episode.
                    index(".")] + "_mirror.vector")

            everything.append("../KDEF/" + sequence + "/" + episode.replace("JPG", "vector
                "))
            everything.append("../KDEF/" + sequence + "/" + episode[:episode.index(".")] +
                "_mirror.vector")

for emotion_id in data:
    if not os.path.exists(emotion_id): os.makedirs(emotion_id)
    for index in data[emotion_id]:
        try:
            shutil.copy2(index, emotion_id)
        except:
            print("Error:", index)

    if not os.path.exists(emotion_id + "_not"): os.makedirs(emotion_id + "_not")
    for index in set(everything) - set(data[emotion_id]):
        try:
            shutil.copy2(index, emotion_id + "_not")
        except:
            print("Error_not:", index)
```

1.1.2 trainers.py

This program is responsible of taking the vectors constructed by 'classification.py' and use different normalization methods. The result of this program is a file named 'emotion.train'. In earlier versions of the program we were using 1vsAll method but since there were major problems with the weight of the natural vectors (having 96% natural vectors vs 4% which was the sum of all other emotions) we decided to use multi classifier. As a result 'trainers.py' produces a file with 8 different type vectors.
Please keep in mind that the part with calculations on the vector will differ between different classifiers.

```
#!/usr/bin/env python3
import os

def distance_between(n1, n2):
        return (abs(n1[0] - n2[0])**2 + abs(n1[1] - n2[1])**2)**0.5

def point_between(n1, n2):
        x = [n1[0], n2[0]]
        y = [n1[1], n2[1]]
        return (abs(x[0] - x[1])/2 + min(x), abs(y[0] - y[1])/2 + min(y))

ck = {
        "Angry"         : 1,
        "Contempt"      : 2,
        "Disgust"       : 3,
        "Fear"          : 4,
        "Happy"         : 5,
        "Sadness"       : 6,
        "Surprise"      : 7,
        "Natural"       : 8 # Other
}

data = {}
```

```
fout = open("emotions.train", "w")
for name in ["Happy", "Sadness", "Surprise", "Angry", "Contempt", "Disgust", "Fear", "
    Natural"]:
        data[str(ck[name])] = []
        for sequence in os.listdir(name):
                data[str(ck[name])].append([])
                for entry in open(name + "/" + sequence):
                        x, y = [float(a) for a in entry.strip().replace("   ", " ").split(
                            " ")]
                        data[str(ck[name])][-1].append((x, y))

for key in data:
        for index in range(len(data[key])):
                i = 1
                vertex = [0.0] + data[key][index]

                fout.write(key)

                left_eye = point_between(vertex[37], vertex[40])
                right_eye = point_between(vertex[43], vertex[46])
                between_eyes = distance_between(left_eye, right_eye)
                nose = point_between(vertex[31], vertex[34])

                for x in range(1, 17 + 1):
                        fout.write(" " + str(i) + ":" + str(distance_between(vertex[x],
                            nose) / between_eyes))
                        i += 1
                for x in range(18, 22 + 1):
                        fout.write(" " + str(i) + ":" + str(distance_between(vertex[x],
                            left_eye) / between_eyes))
                        i += 1
                for x in range(23, 27 + 1):
                        fout.write(" " + str(i) + ":" + str(distance_between(vertex[x],
                            right_eye) / between_eyes))
                        i += 1
                for x in range(32, 36 + 1):
                        fout.write(" " + str(i) + ":" + str(distance_between(vertex[x],
                            nose) / between_eyes))
                        i += 1
                for x in range(37, 42 + 1):
                        fout.write(" " + str(i) + ":" + str(distance_between(vertex[x],
                            left_eye) / between_eyes))
                        i += 1
                for x in range(43, 48 + 1):
                        fout.write(" " + str(i) + ":" + str(distance_between(vertex[x],
                            right_eye) / between_eyes))
                        i += 1
                for x in range(49, 66 + 1):
                        fout.write(" " + str(i) + ":" + str(distance_between(vertex[x],
                            nose) / between_eyes))
                        i += 1
                for x in range(0, 5):
                        fout.write(" " + str(i) + ":" + str(distance_between(vertex[18+x],
                             vertex[27-x]) / between_eyes))
                        i += 1
                for x in range(23, 27 + 1):
                        fout.write(" " + str(i) + ":" + str(distance_between(vertex[x],
                            nose) / between_eyes))
                        i += 1
                for x in range(18, 22 + 1):
```

```
                    fout.write(" " + str(i) + ":" + str(distance_between(vertex[x],
                        nose) / between_eyes))
                    i += 1

                fout.write(" " + str(i) + ":" + str(distance_between(vertex[53], vertex
                    [57]) / between_eyes))
                i += 1
                fout.write(" " + str(i) + ":" + str(distance_between(vertex[52], vertex
                    [58]) / between_eyes))
                i += 1
                fout.write(" " + str(i) + ":" + str(distance_between(vertex[51], vertex
                    [59]) / between_eyes))
                i += 1
                fout.write(" " + str(i) + ":" + str(distance_between(vertex[49], vertex
                    [55]) / between_eyes))
                i += 1
                fout.write(" " + str(i) + ":" + str(distance_between(vertex[50], vertex
                    [54]) / between_eyes))
                i += 1
                fout.write(" " + str(i) + ":" + str(distance_between(vertex[60], vertex
                    [56]) / between_eyes))
                i += 1
                fout.write(" " + str(i) + ":" + str(distance_between(vertex[61], vertex
                    [66]) / between_eyes))
                i += 1
                fout.write(" " + str(i) + ":" + str(distance_between(vertex[62], vertex
                    [65]) / between_eyes))
                i += 1
                fout.write(" " + str(i) + ":" + str(distance_between(vertex[63], vertex
                    [64]) / between_eyes))

                fout.write("\n")
os.system("cp ./emotions.train ./pca_archive_data.txt")
```

1.1.3 pca_training.py

This program takes 'pca_archive_data.txt' which is just a copy of 'emotion.train' and constructs a PCA. Then is vector from the file is being projected over this same PCA and the result is being filtered so that only points with more then a certain percent energy will be taken. As a result some classifiers have 10 others 16,18 and so on. All of this was done so we can decide if more data will make it too specific or too little data would make it too general.
The program also makes sure to store mu, sigma and wt in files as they are crucial for later on reproducing the PCA without the need to build it from scratch.

```
#!/usr/bin/env python3
import random, os
import numpy as np
from matplotlib.mlab import PCA

data = []
for line in open("pca_archive_data.txt"):
        data.append([])
        for el in line[2:].strip().split(" "):
                data[-1].append(float(el[el.index(":")+1:]))
        if len(data[-1]) != 86: data.remove(data[-1])

results = PCA(np.array(data))

archive = open("pca_archive_wt.txt", "w")
for v in results.Wt: archive.write(",".join([str(float(x)) for x in v]) + "\n")
archive.close()
```

```
archive = open("pca_archive_mu.txt", "w")
archive.write(",".join([str(float(x)) for x in results.mu]) + "\n")
archive.close()

archive = open("pca_archive_sigma.txt", "w")
archive.write(",".join([str(float(x)) for x in results.sigma]) + "\n")
archive.close()

fout = open("emotions.train.pca", "w")
for line in open("emotions.train"):
        temp = []
        for el in line[2:].strip().split(" "):
                temp.append(float(el[el.index(":")+1:]))
        fout.write(line[:2] + " ".join([str(str(i+1) + ":" + str(index)) for i, index in
            enumerate(results.project(np.array(temp), 0.01))]) + "\n")
fout.close()
```

1.1.4 easy.py

Because by default easy.py doesn't provide prediction values I had to change it a bit by manually adding '-b 1' flag in the training and later on in the scaling. This makes sure when we provide a vector for testing it will not only tell as which one is closer but how close and do the same for the rest of the values in the model.

```
# easy.py training
65  cmd = '{0} -b 1 -c {1} -g {2} "{3}" "{4}"'.format(svmtrain_exe,c,g,scaled_file,
    model_file)
66  print('Training...')
67  Popen(cmd, shell = True, stdout = PIPE).communicate()

# easy.py prediction
75  cmd = '{0} -b 1 "{1}" "{2}" "{3}"'.format(svmpredict_exe, scaled_test_file, model_file
    , predict_test_file)
76  print('Testing...')
77  Popen(cmd, shell = True).communicate()
```

1.1.5 allinone.py

This program is responsible for checking and displaying the prediction of an image by using a emotion.train.pca.model and emotion.train.pca.range and all of the pca_archive files.
It it important that the formula for normalization in trainers.py is the here as otherwise it will produce different vector and from there the prediction would be wrong.

```
#!/usr/bin/env python3
import os, sys
import numpy as np

ck = {
        1 : "Angry",
        2 : "Contempt",
        3 : "Disgust",
        4 : "Fear",
        5 : "Happy",
        6 : "Sadness",
        7 : "Surprise",
        8 : "Natural/Other"
}

def distance_between(n1, n2):
        return (abs(n1[0] - n2[0])**2 + abs(n1[1] - n2[1])**2)**0.5
```

```
def point_between(n1, n2):
        x = [n1[0], n2[0]]
        y = [n1[1], n2[1]]
        return (abs(x[0] - x[1])/2 + min(x), abs(y[0] - y[1])/2 + min(y))

def fatal(err):
        print(err)
        sys.exit(0)

def v2test(vertex):
        container = []
        vertex = [0.0] + vertex

        left_eye                = point_between(vertex[37], vertex[40])
        right_eye               = point_between(vertex[43], vertex[46])
        between_eyes    = distance_between(left_eye, right_eye)
        nose                    = point_between(vertex[31], vertex[34])

        for x in range(1, 17 + 1):  container.append(distance_between(vertex[x], nose
                ) / between_eyes)
        for x in range(18, 22 + 1): container.append(distance_between(vertex[x], left_eye
                ) / between_eyes)
        for x in range(23, 27 + 1): container.append(distance_between(vertex[x], right_eye
                ) / between_eyes)
        for x in range(32, 36 + 1):     container.append(distance_between(vertex[x], nose
                        ) / between_eyes)
        for x in range(37, 42 + 1):     container.append(distance_between(vertex[x],
           left_eye    ) / between_eyes)
        for x in range(43, 48 + 1):     container.append(distance_between(vertex[x],
           right_eye   ) / between_eyes)
        for x in range(49, 66 + 1):     container.append(distance_between(vertex[x], nose
                        ) / between_eyes)
        for x in range(0, 5):           container.append(distance_between(vertex[18+x],
           vertex[27-x]) / between_eyes)
        for x in range(23, 27 + 1): container.append(distance_between(vertex[x], nose
                ) / between_eyes)
        for x in range(18, 22 + 1):     container.append(distance_between(vertex[x], nose
                        ) / between_eyes)

        container.append(distance_between(vertex[53], vertex[57]) / between_eyes)
        container.append(distance_between(vertex[52], vertex[58]) / between_eyes)
        container.append(distance_between(vertex[51], vertex[59]) / between_eyes)
        container.append(distance_between(vertex[49], vertex[55]) / between_eyes)
        container.append(distance_between(vertex[50], vertex[54]) / between_eyes)
        container.append(distance_between(vertex[60], vertex[56]) / between_eyes)
        container.append(distance_between(vertex[61], vertex[66]) / between_eyes)
        container.append(distance_between(vertex[62], vertex[65]) / between_eyes)
        container.append(distance_between(vertex[63], vertex[64]) / between_eyes)

        return container

if len(sys.argv) < 2: fatal("Usage: ./check_emotion.py [image_name | image_folder] [-o
    output_file_name] [-v save_vectors] [-p save_predict] [-s save_scale] [-a save_all]")
if not os.path.isfile(sys.argv[1]): fatal("The supplied image/folder is not present")
# check if the argv is a folder. In case it is folder go through all of them and save the
    output in a file

data = []

os.system("./face_tracker " + sys.argv[1])
```

```
clean_name = sys.argv[1][:sys.argv[1].index(".")] if "." in sys.argv[1] else sys.argv[1]
for ext in [".vector", "_mirror.vector"][:1]:
        if os.path.isfile(clean_name + ext):
                data.append(v2test([(float(x.strip().split("   ")[0]), float(x.strip().
                    split("   ")[1])) for x in open(clean_name + ext)]))


if not os.path.isfile("pca_archive_wt.txt"): fatal("\"pca_archive_wt.txt\" is missing")
wt_v = [[float(x1) for x1 in x.split(",")] for x in open("pca_archive_wt.txt")]
if not os.path.isfile("pca_archive_mu.txt"): fatal("\"pca_archive_mu.txt\" is missing")
mu_v = [[float(x1) for x1 in x.split(",")] for x in open("pca_archive_mu.txt")][0]
if not os.path.isfile("pca_archive_sigma.txt"): fatal("\"pca_archive_sigma.txt\" is
    missing")
sigma_v = [[float(x1) for x1 in x.split(",")] for x in open("pca_archive_sigma.txt")][0]


fout = open(clean_name + ".pca", "w")
for el in data:
        fout.write("1")
        for wt_id in range(10):
                fout.write(" " + str(wt_id+1) + ":" + str(np.dot(np.array(wt_v[wt_id]), (
                    np.array(el) - np.array(mu_v)) / np.array(sigma_v))))
        fout.write("\n")
fout.close()

if not os.path.isfile("emotions.train.pca.range"): fatal("\"emotions.train.pca.range\" is
    missing")
os.system("./svm-scale -r emotions.train.pca.range \"" + clean_name + ".pca\" > \"" +
    clean_name + ".pca.scale\"")
if not os.path.isfile("emotions.train.pca.model"): fatal("\"emotions.train.pca.model\" is
    missing")
os.system("./svm-predict -b 1 \"" + clean_name + ".pca.scale\" emotions.train.pca.model \"
    " + clean_name + ".pca.predict\" > /dev/null")

if not os.path.isfile(clean_name + ".pca.predict"): fatal("\"" + clean_name + ".pca.
    predict\" is missing")
predict = [line.strip().split(" ") for line in open(clean_name + ".pca.predict")]
for ind in predict[1:]:
        result = {}
        for i, em in enumerate(predict[0][1:]):
                if not ck[int(em)] in result: result[ck[int(em)]] = 0.0
                result[ck[int(em)]] = round(float(ind[i+1])*100, 2)
        label = ""
        for r in sorted(result.items(), reverse=True, key=lambda x: x[1]):
                label += r[0] + " - " + str(r[1]) + "%  "
        print(label)

# put some flags above to mention if we want to save some of there files
os.system("rm " + clean_name + ".vector")
os.system("rm " + clean_name + "_mirror.vector")
os.system("rm " + clean_name + ".pca")
os.system("rm " + clean_name + ".pca.scale")
os.system("rm " + clean_name + ".pca.predict")
```

1.2 Usage

The rotation for training a classifier everything from start to finish would be:

```
./classification.py
./trainers.py
```

```
./pca_training.py
./easy.py emotion.train.pca
```

The rotation for predicting the emotion of an image would be:

```
./allinone image.png
```