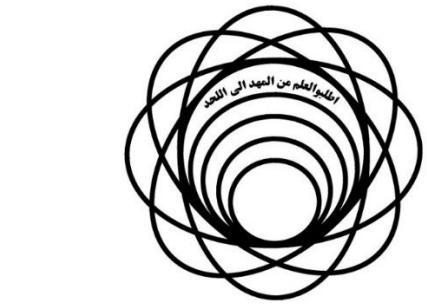

دانشکده فنی و حرفه ای شهید شمسی پور

سیستم تشخیص لبخند

مستند فنی پروژه

استاد : محمد صادق مجتبی فر

دانشجو : پارسا مرادی

رشته : کامپیوتر گرایش نرم افزار

خرداد ماه 1403


بسم الله الرحمن الرحیم

فهرست

مقدمه

چکیده

این پروژه به منظور توسعه یک سیستم تشخیص لبخند با استفاده از مدل‌های یادگیری ماشین و تکنیک‌های پردازش تصویر ایجاد شده است. هدف اصلی این پروژه تشخیص لبخند در تصاویر افراد است. در آینده، این سیستم به تشخیص احساسات گسترده‌تری مانند شادی، ناراحتی و غیره توسعه خواهد یافت .

هدف پروژه

هدف اصلی این پروژه ایجاد یک مدل هوش مصنوعی است که قادر به تشخیص لبخند در تصاویر باشد همچنین، یک وب سایت با استفاده از FastAPI طراحی شده است تا کاربران بتوانند تصاویر خود را آپلود کرده و نتایج تشخیص لبخند را مشاهده کنند .

نتایج و دستاوردها

دقت مدل در تشخیص لبخند به میزان قابل قبولی است.

وب سایتی ایجاد شد که کاربران می‌توانند تصاویر خود را آپلود کرده و نتیجه تشخیص لبخند را مشاهده کنند .

چالش ها

جمع‌آوری و آماده‌سازی داده‌ها

تنظیمات بهینه برای مدل KNN

پیاده‌سازی صحیح وب سایت با FastAPI

مسیرهای آینده

گسترش سیستم به تشخیص انواع احساسات

بهبود دقت مدل با استفاده از تکنیک‌های پیشرفته‌تر یادگیری ماشین و یادگیری عمیق

توسعه رابط کاربری وب سایت برای تجربه کاربری بهتر

تکنولوژی‌ها و ابزارهای استفاده شده

Python: زبان برنامه‌نویسی اصلی برای توسعه مدل و پیاده‌سازی وب سایت

OpenCV: برای پردازش تصویر

NumPy: برای عملیات‌های عددی

scikit-learn: برای پیاده‌سازی مدل KNN

joblib: برای ذخیره و بارگذاری مدل

FastAPI: برای پیاده‌سازی وب سایت

طراحی و تعلیم مدل

```
import cv2
import glob
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
import joblib

Data_list = []
Label_list = []

for address in glob.glob(r"C:\Users\R369\Desktop\final_project\smile_dataset\*\*"):
    img = cv2.imread(address)
    img = cv2.resize(img,(32,32))
    img = img / 255.0
    img = img.flatten()
    Data_list.append(img)
    p = address.split("\\")[-1].split(".")[0]
    Label_list.append(p)

Data_list = np.array(Data_list)
x_train , x_test , y_train , y_test =
train_test_split(Data_list,Label_list,test_size=0.2)

clf = KNeighborsClassifier(9)
clf.fit(x_train,y_train)
joblib.dump(clf,"smile.z")
pre = clf.predict(x_test)
acc = accuracy_score(y_test,pre)
print(acc*100)
```

مراحل پروژه :

1. جمع‌آوری و آماده‌سازی داده‌ها

تصاویر مختلفی که شامل لبخند و عدم لبخند بودند جمع‌آوری شدند. این تصاویر سپس به اندازه‌های ثابت تغییر اندازه داده شدند و برای استفاده در مدل آمادەسازی شدند.

```
img = cv2.imread(address)
```

2. پیش‌پردازش و نرمال‌سازی تصاویر

با استفاده از کتابخانه OpenCV، تصاویر به اندازه 32x32 تغییر اندازه داده شدند و نرمال‌سازی شدند تا مقادیر پیکسل‌ها بین 0 و 1 قرار بگیرند.

```
img = cv2.resize(img,(32,32))
```

3. تقسیم‌بندی داده‌ها

داده‌ها به دو مجموعه آموزشی و تست تقسیم شدند تا مدل بتواند بر اساس این داده‌ها آموزش ببیند و ارزیابی شود.

```
img = img / 255.0
```

4. آموزش مدل

یک مدل K-Nearest Neighbors (KNN) با استفاده از کتابخانه scikit-learn آموزش داده شد. این مدل با استفاده از داده‌های آموزشی تصاویر را طبقه‌بندی می‌کند.

```
clf = KNeighborsClassifier(9)
```

5. ذخیره مدل

مدل آموزش دیده با استفاده از کتابخانه joblib ذخیره شد تا در مراحل بعدی برای پیش‌بینی استفاده شود.

```
joblib.dump(clf,"smile.z")
```

6. ارزیابی مدل

مدل با استفاده از داده‌های تست ارزیابی شد و دقت آن محاسبه گردید. نتایج نشان داد که مدل با دقت قابل قبولی لبخند را تشخیص می‌دهد.

```
pre = clf.predict(x_test)
acc = accuracy_score(y_test,pre)
print(acc*100)
```

خواندن و استفاده از مدل

این اسکریپت به منظور لود کردن مدل آموزش‌دیده KNN و استفاده از آن برای پیش‌بینی لبخند در تصاویر جدید طراحی شده است. مدل KNN قبلاً با استفاده از داده‌های آموزشی آموزش دیده و ذخیره شده است. اکنون می‌توانیم از این مدل برای طبقه‌بندی تصاویر جدید در وبسایتمان استفاده کنیم .

```
import cv2
from joblib import load
import glob
import numpy as np

clf = load("smile.z")

def load_pic(item):
    img = cv2.imread(item)
    img_r = cv2.resize(img,(32,32))
    img_r = img_r / 255
    img_r = img_r.flatten()
    img_r = np.array([img_r])
    label = clf.predict(img_r)[0]

    return label
```

لود کردن مدل KNN

مدل آموزش‌دیده KNN را از فایل ذخیره شده لود می‌کنیم

```
clf = load("smile.z")
```

تعریف تابع برای لود و پیش‌بینی تصاویر

تابعی به نام load_pic تعریف می‌کنیم که تصویر را لود کرده، پیش‌پردازش کرده و برچسب آن را با استفاده از مدل KNN پیش‌بینی می‌کند

```
def load_pic(item):
    img = cv2.imread(item)
    img_r = cv2.resize(img,(32,32))
    img_r = img_r / 255
    img_r = img_r.flatten()
    img_r = np.array([img_r])
    label = clf.predict(img_r)[0]

    return label
```

توضیح عملکرد تابع load_pic

خواندن تصویر:

با استفاده از cv2.imread(item) تصویر را از مسیر مشخص شده می‌خوانیم.

تغییر اندازه تصویر:

با استفاده از cv2.resize(img, (32, 32)) اندازه تصویر را به 32x32 پیکسل تغییر می‌دهیم تا با داده‌های آموزشی مطابقت داشته باشد.

نرمال‌سازی تصویر:

تصویر را با تقسیم بر 255 نرمال‌سازی می‌کنیم تا مقادیر پیکسل‌ها بین 0 و 1 قرار بگیرند.

تبدیل تصویر به آرایه تک‌بعدی:

با استفاده از flatten() تصویر را به آرایه‌ای تک‌بعدی تبدیل می‌کنیم.

ایجاد آرایه با بعد اضافه:

با استفاده از np.array([img_r]) آرایه‌ای با بعد اضافه برای ورودی به مدل ایجاد می‌کنیم.

پیش‌بینی برچسب تصویر:

با استفاده از clf.predict(img_r)[0] برچسب تصویر را پیش‌بینی می‌کنیم.

استفاده از مدل در بستر وب

1. وارد کردن کتابخانه‌ها و وابستگی‌ها

ابتدا کتابخانه‌ها و وابستگی‌های مورد نیاز را وارد می‌کنیم

```
import os
import pathlib
import random
import string
from fastapi import FastAPI, File, HTTPException, Request, UploadFile, status
from fastapi.responses import HTMLResponse, StreamingResponse
from fastapi.staticfiles import StaticFiles
from core.jinja import jinja
from core import ai
```

2. ایجاد نمونه‌ای از FastAPI

یک نمونه از کلاس FastAPI ایجاد می‌کنیم که به عنوان برنامه اصلی API عمل می‌کند

```
app = FastAPI(title="HowRU")
```

3. افزودن فایل‌های استاتیک

مسیر مربوط به فایل‌های استاتیک را مشخص می‌کنیم

```
app.mount("/static", StaticFiles(directory="static"), name="static")
```

4. روت‌ها

مسیر اصلی برای بررسی وضعیت API

این مسیر یک پاسخ HTML بازمی‌گرداند که نشان می‌دهد API کار می‌کند یا نه

```
@app.get(
    path="/",
    status_code=status.HTTP_200_OK,
    description="This endpoint returns an HTML response with a message telling
you if the api is working or not",
    tags=["API"],
    name="API checker",
    summary="Check if the API is live or not, just it",
    response_class=HTMLResponse,
)
async def api_check(request: Request):
    message = "FACE"
    context = {"message": message}
    return jinja.response(request=request, name="form.html", context=context)
```

مسیر برای پردازش تصویر آپلود شده

این مسیر تصویر آپلود شده را دریافت کرده و آن را پردازش می‌کند و نتیجه را به کاربر بازمی‌گرداند

```
@app.post(
    path="/process",
    description="Upload picture and prepare that for pass to AI model for
processing and send result to client",
    status_code=status.HTTP_200_OK,
    summary="Main job of whole application",
    tags=["PROCESS", "API"],
    name="process",
)
async def upload_picture(request: Request, file: UploadFile = File(...)):
    main_path = os.path.dirname(os.path.abspath(__file__))
    upload_path = f"{main_path}/uploads"

    file_extension = file.filename.split(".")[-1].lower()

    if file_extension not in ["png", "jpg"]:
        raise HTTPException(
            status_code=400,
            detail="Unsupported file format. Only PNG and JPG are allowed.",
        )
```

```
    file_name = generate_random_string(10) + f".{file_extension}"

    uploaded_file_path = f"{upload_path}/{file_name}"

    with open(uploaded_file_path, "wb") as file_temp:
        file_temp.write(await file.read())



    result = ai.load_pic(uploaded_file_path)


    message = result

    context = {"message": message, "file_name": file_name}

    return jinja.response(request=request, name="result.html", context=context)
```

مسیر برای نمایش فایل

این مسیر فایل آپلود شده را به کاربر نمایش می‌دهد

```
@app.get("/file/{file_name}")
async def show_file(file_name: str):
    main_path = os.path.dirname(os.path.abspath(__file__))
    upload_path = f"{main_path}/uploads"

    file_path = pathlib.Path(f"{upload_path}/{file_name}")

    if not file_path.exists() or not file_path.is_file():
        return {"error": "File not found"}
    return StreamingResponse(
        file_iterator(file_path),
        media_type="application/octet-stream",
        headers={"Content-Disposition": f"attachment; filename={file_name}"},
    )
```

اجرای برنامه

1- برای اجرای برنامه در قدم اول باید تمامی کتابخانه های مورد نیاز را نصب کنید برای اینکار باید دستوری که در فایل pip.ps1 در پوشه commands را کپی کنید

2- حال با زدن کلید ترکیبی ctrl+` ترمینال را باز میکنید و متنی که کپی کردید را وارد میکنید و دکمه enter را میزنید

3- حال باید دستور اجزرای برنامه که در فایل uvicorn.ps1 در پوشه commands را کپی کنید و مانند دستور قبل آن را در ترمینال وارد میکنید و دکمه enter را میزنید

4- آدرس http://127.0.0.1:8000 را در مرورگر خود وارد میکنید و وارد وب سایت می شوید